


## Business Plan

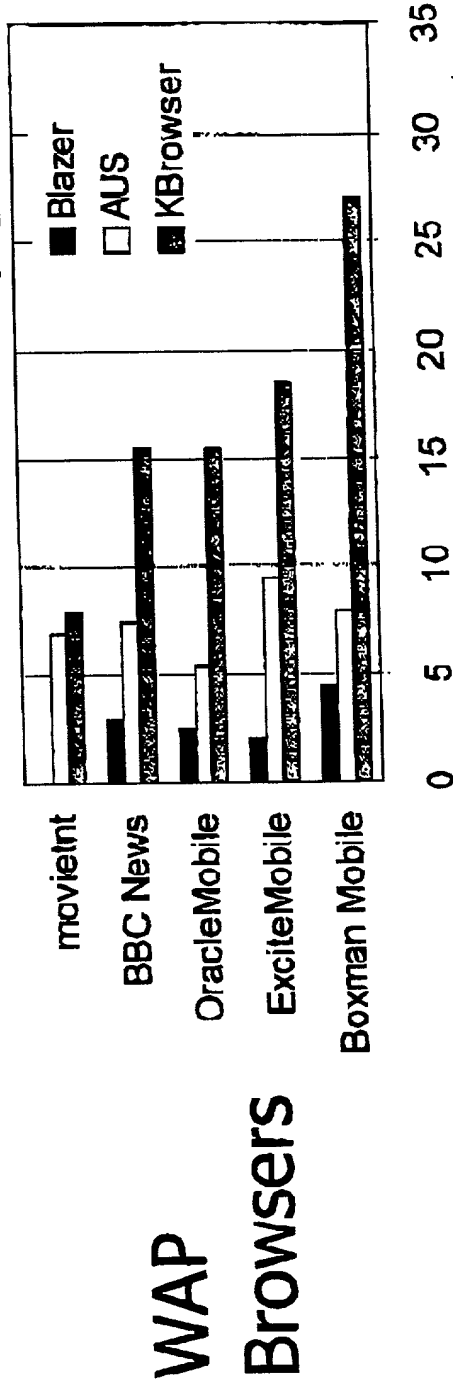
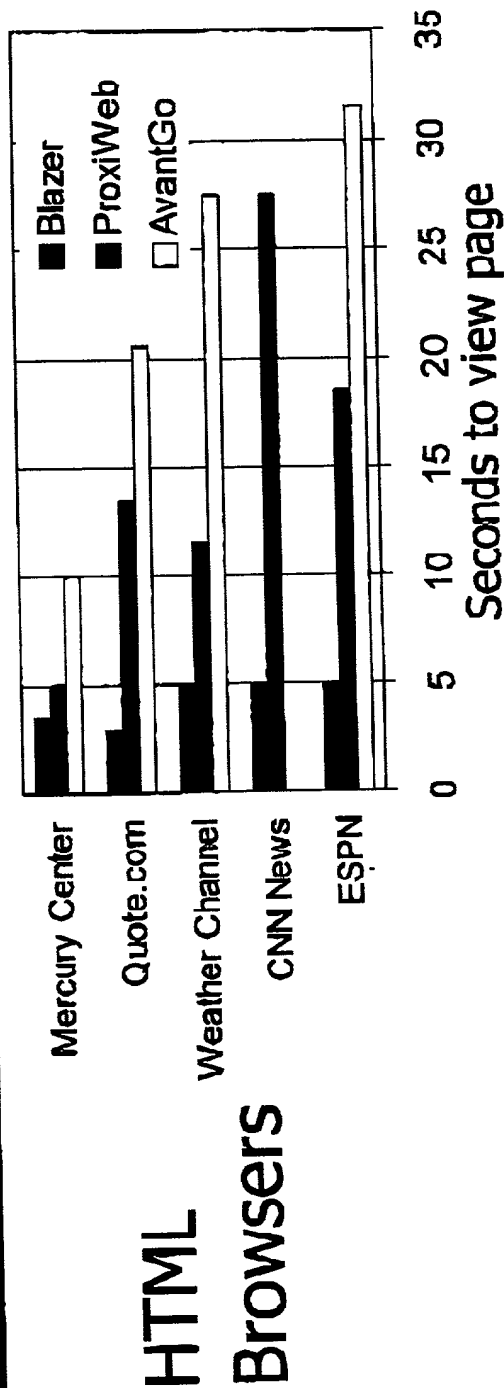
	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2
--	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---

FOR FURTHER INFORMATION

 **BlueLark Systems, Inc.**

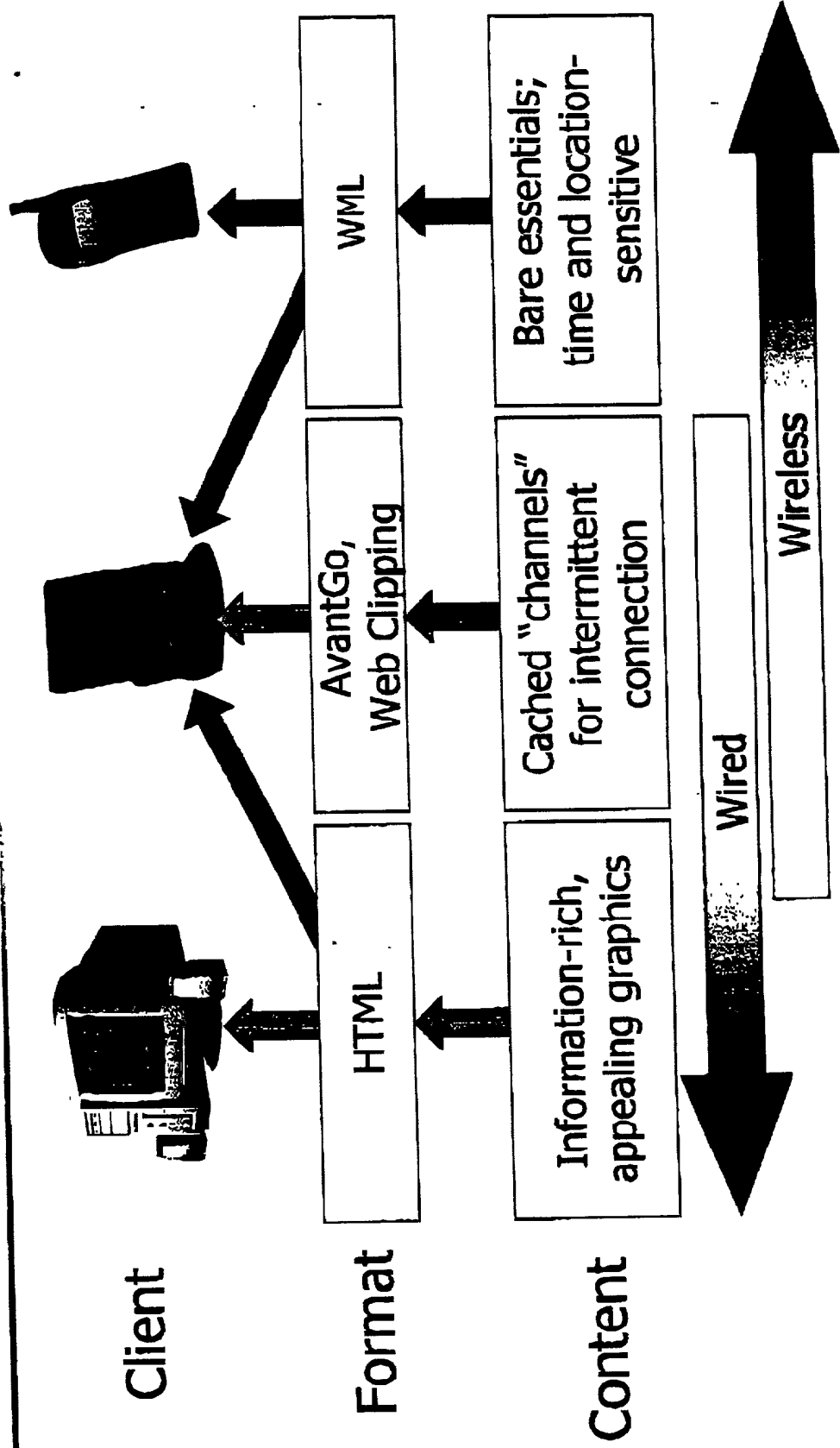
---

# Speed Comparison



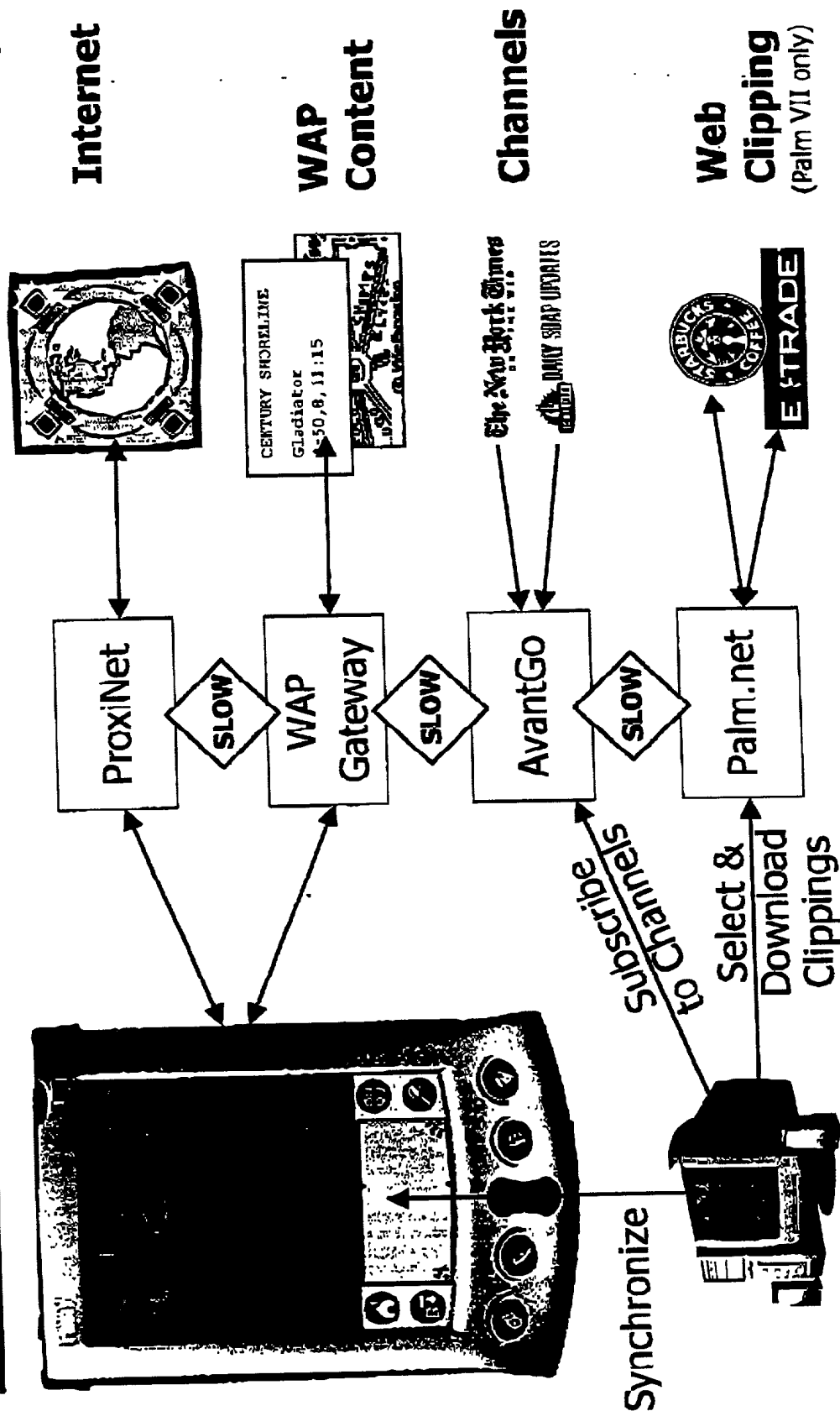
Tests simulated wireless performance using a wireline modem set at 19.2kbps

# Content Converging on PDAs

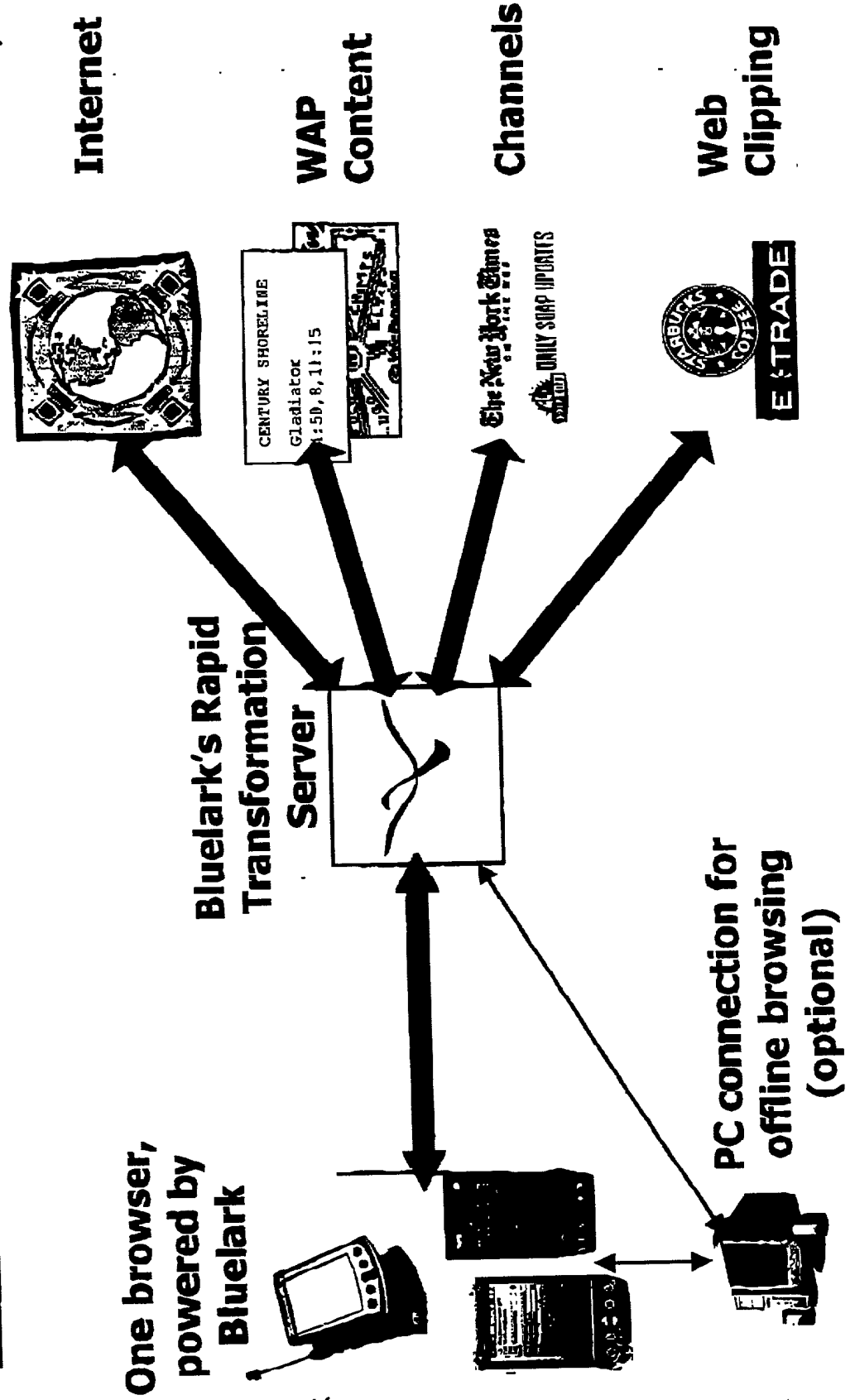




# Current Solution



# BlueLark Solution



# Why The Difference in Speed?

## Other Systems

Proxy Server waits  
for entire page

Browser waits  
for entire page

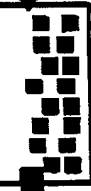
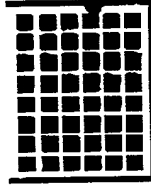
THEN



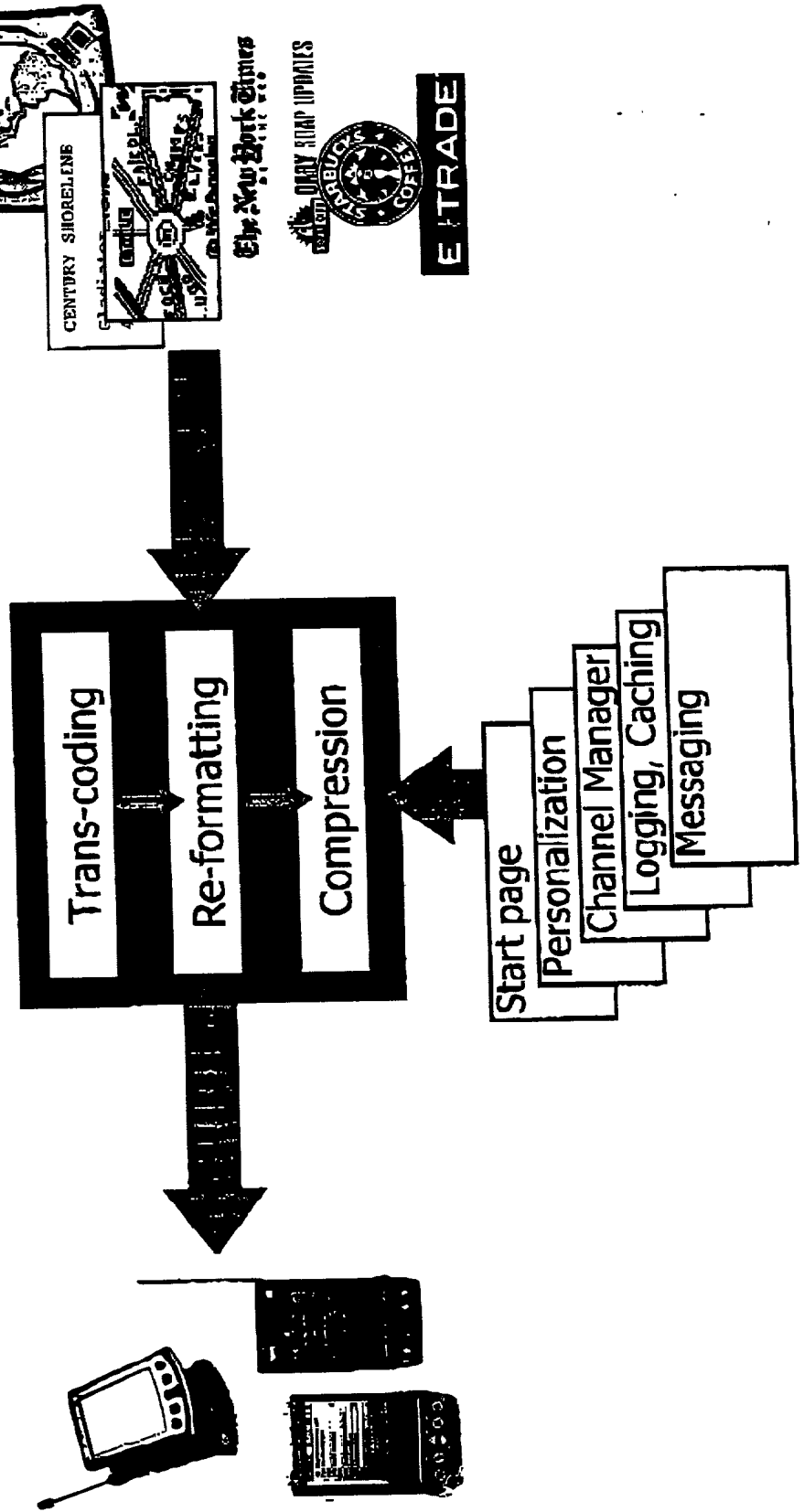
## BlueLark

Proxy Server  
streams content,  
transforming on  
the fly

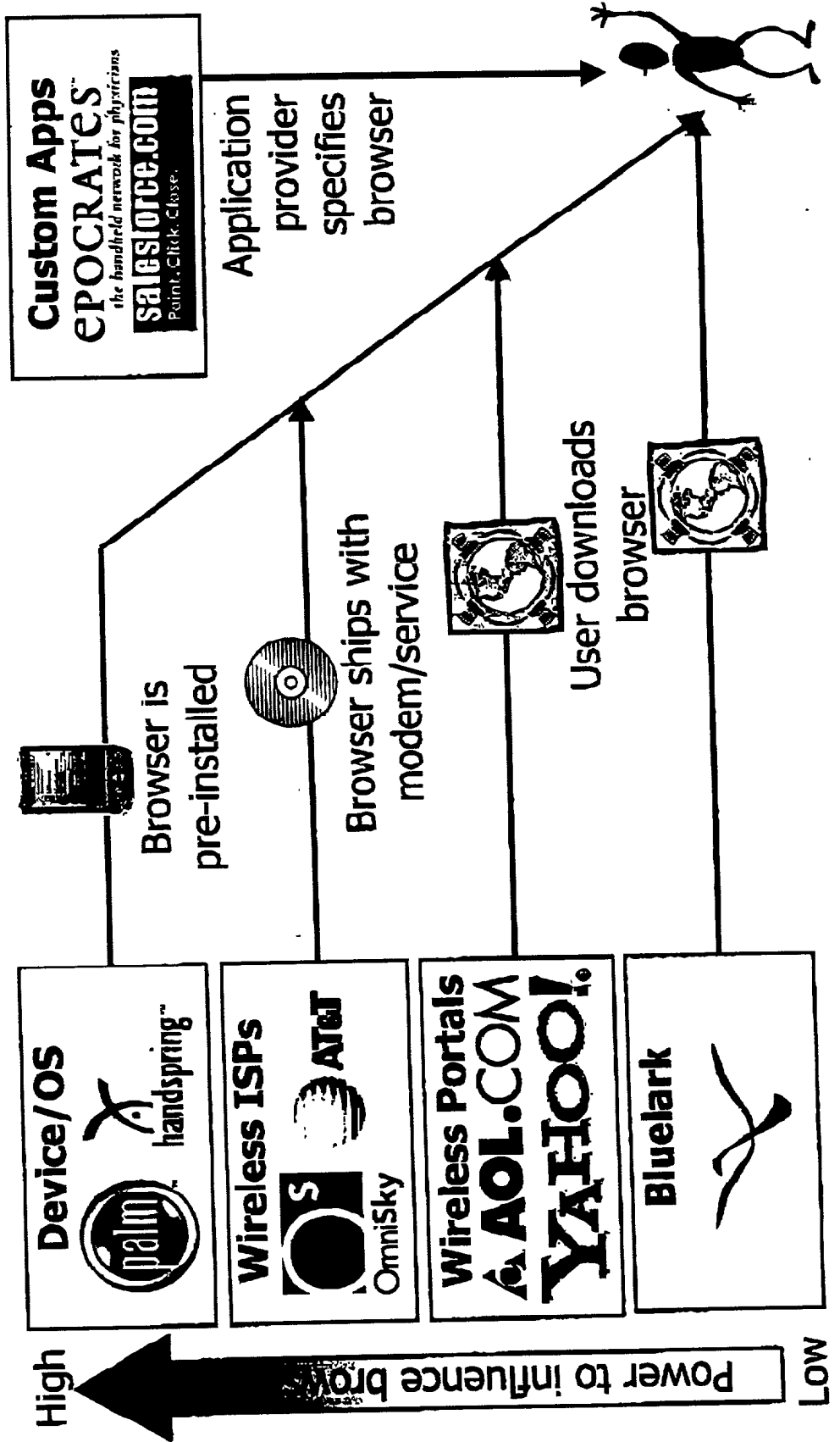
Browser  
renders  
progressively



# What Does the RTS Do?



# Distribution to the End-User



# Revenue Sources

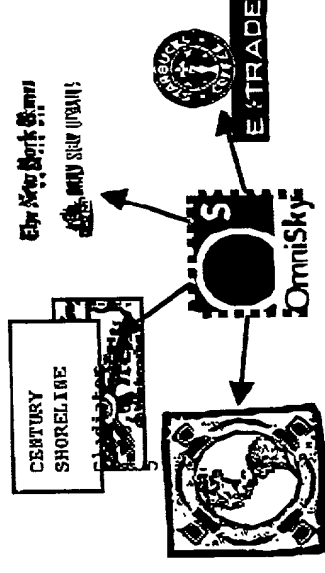
1. License technology to wireless ISPs
2. License customized browser to portals
3. Channel new users to portals
4. Customized applications and consulting
5. Platform for plug-in applications
6. Service & updates
7. Additional services to end-users

Objective: Become the standard Internet access platform for PDAs

# 1. License to Wireless ISPs

## Benefits to ISPs:

- Give customers a better service
  - Fast access – "Think it, do it!"
  - The whole web
  - Channel manager
- Retain portal role
  - Fixed start page
  - Avoid being circumvented by users installing a better browser



## 2. License to Portals

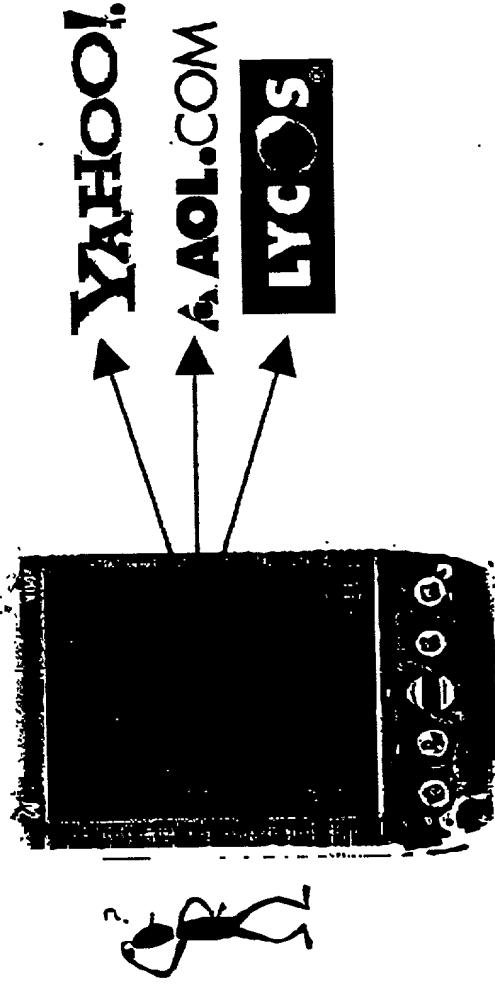
### Benefits to portals:

- Be the AOL of wireless computing
  - Start-page is "hard-coded"
  - Move functions to PDA UI (e.g. Search)
- Use proxy-server connection to push personalized, location-sensitive, time sensitive content



## 3. Channel New Users

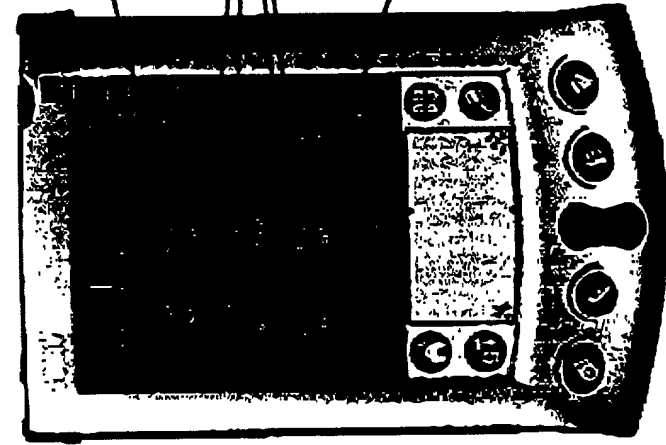
1. User downloads browser from Blueark
2. Browser launches with Blueark start page
3. User chooses a new start page from menu of Blueark partners
4. Blueark logs browser use and charges portal



- Portals pay Blueark to be start-page partners
  - Listing fee
  - Per user fee
  - Click-through (per use) fee

## 4. Custom applications

*ePocrates is the leading provider of real-time medical information delivered to the point of care*



ePocrates qRx  
drug reference

Proprietary web-based medical information  
sources (e.g. New England Journal of  
Medicine), delivered by customized  
versions of the BlueLark browser

Standalone version of  
the BlueLark browser

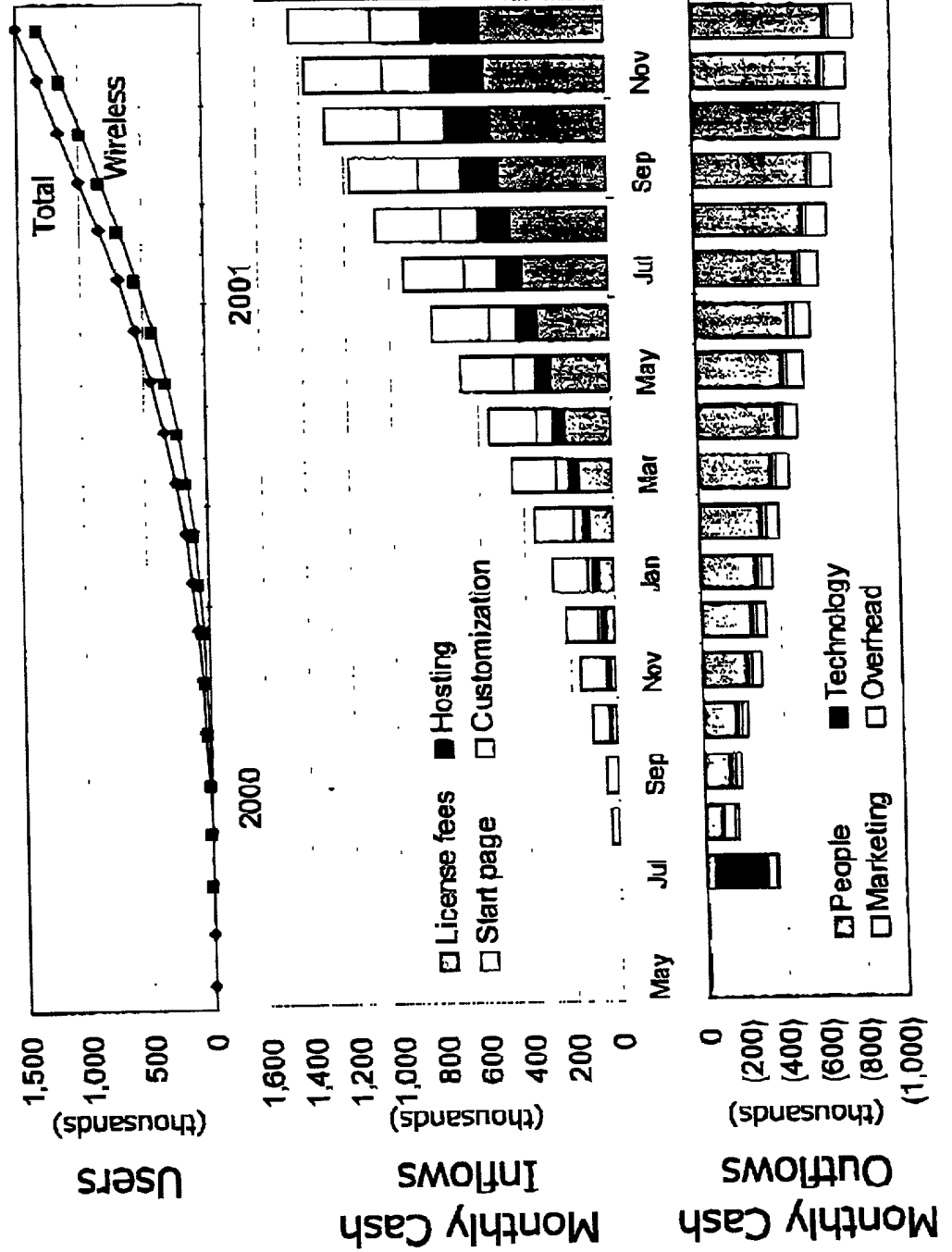
- BlueLark gives ePocrates full control over the appearance, branding and delivery of information to physicians

# Lots of Custom Applications

---

- Sales Force Automation (Siebel, Salesforce.com, ...)
- Facilities Management (Aperture, ...)
- Delivery Tracking and Logging (FedEx, ...)
- Fleet Management (SuperShuttle, ...)
- Construction Management (Cephren, BidCom, ...)
- Practice Management (McKesson/HBOC, ...)
- Group Scheduling (Evite.com, ...)
- Yellow Pages (AirFlash, Vindigo, Scoot, ...)
- Instant Payments (PayPal, ...)
- etc...

# Cash Flow Projections



# Assumptions

- Market Size (IDC)
  - 3 million U.S. wireless PDA users by 12/01
  - 17 million by 12/03
- License fees:
  - \$1-\$20 per user
- Recurring hosting & service fees:
  - \$0.10-\$0.30 per user per month
- Start page partners:
  - \$5+ per user plus hosting fees
- Customization & consulting:
  - \$150/hour

# Exit Strategy

- Acquisition
  - Excellent match with PumaTech, Riverbed (Aether), Spyglass, Palm, Handspring, OmniSky, Yahoo!, many others
- IPO
  - Opportunity to become industry-standard platform

# PDA Browser Competition

Company	Product	HTML	WML	Web Clipping	Offline Channels	Palm III/V	Palm VII	Other Devices	Images	Uses Proxy	Security	Strategic Focus
BlueLark	Blazer	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Certicom Jul-00	Software that enables fast, easy wireless PDA Internet use
AvantGo	AvantGo	Yes	No	No	Yes	Yes	No	Win CE	Yes	Yes	Certicom	Channel portal
Palm, others	Web Clippings	No	DP only	Yes	No	Soon	Yes	No	Cached only	Yes	Yes	Channel portal, add value to platform
ProxiNet (Puma Tech)	ProxiWeb	Yes	No	No	No	Yes	No	No	Squished	No	3DES/SSL	Mobile device synchronization, development tools, personalized alerts
Qualcomm/Kyocera	pdQ Browser	Yes	No	No	No	Yes	No	No	No	No	?	Unclear following sale to Kyocera
4thPass	KBrowser	No	Yes	No	No	WML only	Clipping only	None yet	WBMP only	Yes	?	Java software development tools. Minlu.com portal under development
Neomar	Neomar Microbrowser	?	Yes	?	No	Yes	?	RIM (eLink)	?	No	?	Mobile portal plus end-to-end WAP content delivery technology
AUSystem	AUS WAP Browser	No	Yes	No	No	IIIx and V	No	Ericsson phones +	No	No	?	Data communications software, with focus on mobile phones
The Edge Consultants	WAPMan	No	Yes	No	No	Yes	No	8598/NT, phones	No	No	?	Unclear. Singapore-based consulting group
Pico	Pico Microbrowser	Yes	No	No	No	?	?	Phones	?	No	?	Mobile phone software (browser & PIM applications)
Microsoft	Pocket IE	Yes	No	No	Yes	No	No	Win CE	Yes	No	Yes	OS sales

**Our system is the fastest by a wide margin, and the only one that reads the full range of Internet content. But more competition is coming...**

# Risks

---

- Technology risks fairly low
  - Should scale well, but not yet tested
- Market risks significant; will be resolved soon
  - Sales cycle may be longer than we expect
  - Adoption of wireless modems uncertain
    - Pricing & reliability are key
  - Need to move FAST!
  - Competition may get rough
    - What are Palm's plans?



# Execution Challenges

---

- Grab share
  - Requires strong Biz Dev and Marketing
- Grow the organization fast
  - Maintain speed, flexibility and culture
- Scale ahead of the curve
  - Server & bandwidth infrastructure
  - Maintain user support
- Find all the customers
  - Don't let segments slip away

# Core Team

---

## **Simon Mawby**

B.A. & M.A. Oxford; MBA Stanford

Most recently Director of Product Development at ePocrates, Inc.  
Background in Management Consulting, Sales and Sales Management

## **Jon Kleid**

B.S. Computer Science, Stanford

Managed client-side development for ePocrates' qRx  
Software Engineer at Sun Microsystems, Connectix and ForGolf

## **Ben Strong**

B.S. Computer Science, Stanford

Managed Server-Side Development for ePocrates' qRx  
Software Engineer at SpaceMachine and iCarz

## **Vivek Patel**

B.S. Computer Science, Stanford

Managed database development for ePocrates' qRx  
Software Engineer at MediaOne

## Server Source Code

[illegible]

```

// MALConduitManager.cpp
#include "MALConduitManager.h"
#include "ace/OS.h"
#include "BLUtils.h"
#include "conduits/MALConduitDefs.cpp"

MALConduitManager* MALConduitManager::sInstanceP = NULL;

MALConduitManager*
MALConduitManager::instance()
{
    if (sInstanceP == NULL)
    {
        sInstanceP = new MALConduitManager();
        return sInstanceP;
    }
}

MALConduitManager::MALConduitManager()
{
}

MALConduitManager::~MALConduitManager()
{
    for (MALDatabaseMap::iterator i = mDatabaseTable.begin();
         i != mDatabaseTable.end();
         i++)
    {
        char *dbName = (char*) (*i).ext_id;
        delete dbName;
    }
}

void
MALConduitManager::initialize()
{
    for (std::vector<MALConduitFactory*>::iterator i = mActiveFactories.begin();
         i != mActiveFactories.end();
         i++)
    {
        (*i)->initialize(this);
    }
}

void
MALConduitManager::finalize()
{
    for (std::vector<MALConduitFactory*>::iterator i = mActiveFactories.begin();
         i != mActiveFactories.end();
         i++)
    {
        (*i)->finalize();
    }
}

MALConduitFactory*
MALConduitManager::activateFactory(const char *name)
{
    for (int i=0; sFactoryDefs[i] != NULL; i++)
    {
        if (ACE_OS::strcmp(name, sFactoryDefs[i]->name()) == 0)
        {
            mActiveFactories.push_back(sFactoryDefs[i]);
            return sFactoryDefs[i];
        }
    }
    return NULL;
}

void
MALConduitManager::registerDatabase(const char *name, MALConduitFactory *factoryP)
{

```

```

    int factoryIndex = 0;
    for (std::vector<MALConduitFactory*>::iterator i = mActiveFactories.begin();
         i != mActiveFactories.end();
         i++, factoryIndex++)
    {
        if (*i == factoryP)
        {
            const char *nameCopy = BLUtils::strdup(name);
            mDatabaseTable.bind(nameCopy, factoryIndex);
            break;
        }
    }

    int
    MALConduitManager::getIndexForDatabase(const char *name)
    {
        ACE_Hash_Map_Entry<const char*, int> *entry;
        if (mDatabaseTable.find(name, entry) == 0)
            return entry->int_id;
        return -1;
    }

    MALConduitDispatcher*
    MALConduitManager::makeDispatcher()
    {
        std::vector<MALConduit*> *conduitsP = new std::vector<MALConduit*>();
        for (std::vector<MALConduitFactory*>::iterator i = mActiveFactories.begin();
             i != mActiveFactories.end();
             i++)
        {
            conduitsP->push_back((*i)->makeConduit());
        }
        return new MALConduitDispatcher(conduitsP, this);
    }

    MALConduitDispatcher::MALConduitDispatcher(std::vector<MALConduit*> *conduitsP,
                                                MALConduitManager *managerP)
    {
        mConduitsP(conduitsP),
        mManagerP(managerP)
    }

    MALConduitDispatcher::MALConduitDispatcher()
    {
        for (std::vector<MALConduit*>::iterator i = mConduitsP->begin();
             i != mConduitsP->end();
             i++)
        {
            MALConduit* conduitP = *i;
            delete conduitP;
        }
        delete mConduitsP;
    }

    MALConduit*
    MALConduitDispatcher::getConduitForDatabase(const char *name)
    {
        int index = mManagerP->getIndexForDatabase(name);
        if (index != -1)
            return (*mConduitsP)[index];
        return NULL;
    }

    void
    MALConduitDispatcher::sendInitialResponses(MALWriter *writerP,

```

BIClientInfo \*infoP;

```
for (std::vector<MALConduit*>::iterator i = mConduitsP->begin();
     i != mConduitsP->end();
     i++)
```

```
    MALConduit* conduitP = *i;
    conduitP->sendInitialResponse(writerP, infoP);
}
```

```
void
MALConduitDispatcher::sendNormalResponses(MALWriter *writerP,
                                           BIClientInfo *infoP)
```

```
{
    for (std::vector<MALConduit*>::iterator i = mConduitsP->begin();
         i != mConduitsP->end();
         i++)
```

```
    {
        MALConduit* conduitP = *i;
        conduitP->sendNormalResponse(writerP, infoP);
    }
}
```

// MALConfig.cpp

```
#include "MALConfig.h"
#include "MALConstants.h"
#include "MALConduitManager.h"
#include "ace/OS.h"
#include "BLUtils.h"
#include "xmlparse.h"
```

```
MALConfig::MALConfig()
: mConduitManagerP(MALConduitManager::instance()),
  mInterface(NULL)
```

```
MALConfig::~MALConfig()
{
    delete mInterface;
}
```

int readConfigFile(char \*fileName)

```
{
    ACE_HANDLE infile = ACE_OS::open (fileName, O_RDONLY);
```

```
    if (infile == ACE_INVALID_HANDLE)
    {
        ACE_ERROR ((LM_ERROR, "%p\n", fileName));
        return -1;
    }
}
```

```
XML_Parser parser = XML_ParserCreate(NULL);
XML_SetUserData(parser, this);
XML_SetElementHandler(parser,
    xmlStartElementHandler,
    xmlEndElementHandler);
```

```
mErrCode = 0;
mCurrentFactoryP = NULL;
mPort = KMALDefaultPort;
mInterface = NULL;
```

```
char buf[kFileBufSize];
int len;
do {
    len = ACE_OS::read(infile, buf, kFileBufSize);
```

if (len == -1)

```
    ACE_ERROR ((LM_ERROR, "Error while reading config file: %s\n", fileName));
    mErrCode = -1;
    break;
```

```
    else if (!XML_Parse(parser, buf, len, len == 0))
```

```
    {
        ACE_ERROR ((LM_ERROR, "Parse Error: %s%d %s\n", fileName,
            XML_GetCurrentLineNumber(parser),
```

XML\_ErrorString(XML\_GetErrorCode(parser))));

```
    mErrCode = -1;
    break;
```

```
    } while (len > 0);
```

```
XML_ParserFree(parser);
ACE_OS::close (infile);
```

return mErrCode;

```
void
MALConfig::xmlStartElementHandler(void *userData,
    const char *name,
    const char **atts)
```

```
{
    MALConfig *configP = (MALConfig*) userData;
```

```
    // server tag
    if (ACE_OS::strcmp(name, "malserver") == 0)
```

```
    {
        for (const char **attP=atts; *attP!=0; attP++)
```

```
        {
            const char *attName = *attP;
            const char *attValue = *++attP;
```

```
            if (strcmp(attName, "port") == 0)
```

```
            {
                configP->mPort = ACE_OS::atoi(attValue);
```

```
            }
            else if (strcmp(attName, "interface") == 0)
```

```
            {
                configP->mInterface = BLUtils::strdup(attValue);
```

```
            }
            else
```

```
            {
                configP->unknownOption(attName);
```

```
            }
        }
    }
    // conduit tag
    else if (strcmp(name, "conduit") == 0)
```

```
    {
        const char *name = NULL;
        for (const char **attP=atts; *attP!=0; attP++)
```

```
        {
            const char *attName = *attP;
            const char *attValue = *++attP;
```

```
            if (strcmp(attName, "name") == 0)
```

```
            {
                name = attValue;
```

```
            }
            else
```

```
            {
                configP->unknownOption(attName);
```

```
            }
        }
    }
    if (name != NULL)
```

```
    {
```

```

    configp->mCurrentFactoryP = configp->mConduitManagerP->activateFactory(name);
    if (configp->mCurrentFactoryP == NULL)
    {
        ACE_ERROR ((LM_ERROR, "Error: Could not activate Conduit: %s\n", name));
        configp->mErrCode = -1;
    }
}

// option tag
else if (strcmp(name, "option") == 0)
{
    const char *name = NULL;
    const char *value = NULL;
    for (const char **attP=atts; *attP!=0; attP++)
    {
        const char *attName = *attP;
        const char *attValue = *(++attP);
        if (strcmp(attName, "name") == 0)
        {
            name = attValue;
        }
        else if (strcmp(attName, "value") == 0)
        {
            value = attValue;
        }
        else
        {
            configp->unknownOption(attName);
        }
    }

    if (name != NULL && value != NULL && configp->mCurrentFactoryP != NULL)
    {
        if (configp->mCurrentFactoryP->setOption(name, value) != 0)
        {
            configp->unknownOption(name);
        }
    }
}

void
MALConfig::xmlEndElementHandler(void *userData,
                                const char *name)
{
    MALConfig *configP = (MALConfig*) userData;
    // should probably try to validate a little more here

    void
    MALConfig::unknownOption(const char *option)
    {
        ACE_ERROR ((LM_ERROR, "Error: Unknown Configuration Option: %s\n", option));
        mErrCode = -1;
    }
    // MALIO.cpp
    #include "MALIO.h"

    void
    MALIO::init(ACE_SOCK_Stream *streamP, AGRewriter *readerP, AGRewriter *writerP)
    {
        AGRewriterInit(readerP, streamP, MALIO::read);
        AGRewriterInit(writerP, streamP, MALIO::write);
    }

    int
    MALIO::read(void *data, void *dst, int len)
    {
        ACE_SOCK_Stream *streamP = (ACE_SOCK_Stream*) data;

```

```

    int readLen = streamP->recv_n((char*)dst, len);
    return readLen;
}

int
MALIO::write(void *data, void *buf, int len)
{
    ACE_SOCK_Stream *streamP = (ACE_SOCK_Stream*) data;

    int writeLen = streamP->send_n((char*)buf, len);
    return writeLen;
}

// MALSession.cpp
#include "MALSession.h"
#include "MALIO.h"
#include "MALConduitManager.h"
#include "MALConduit.h"
#include "MALConstants.h"
#include "AGProtocol.h"
#include "AGDigest.h"

MALSession::MALSession()
{
    mDispatcherP = MALConduitManager::instance()->makeDispatcher();
    mConduitP = NULL;
    mInitialSession = true;

    MALSession::~MALSession()
    {
        delete mDispatcherP;
    }

    // take care of this here so other clients of BLClientInfo don't have to use malloc and
    free
    free(mClientInfo.getUserName());
    free(mClientInfo.getOSName());
    free(mClientInfo.getOSVersion());
    free(mClientInfo.getSerialNumber());
    free(mClientInfo.getLanguage());
    free(mClientInfo.getCharset());
}

int
MALSession::doSession(ACE_SOCK_Stream *streamP)
{
    MALIO::init(streamP, mReader, mWriter);

    if (readHeader() != 0)
        return -1;

    if (readCommands() != 0)
        return -1;

    // write http header
    char* status = "HTTP/1.0 200 OK\r\n\r\n";
    int len = ACE_OS::strlen(status);
    if (streamP->send_n(status, len) != len)
        return -1;

    int err = sendResponse();
    return err;
}

int
MALSession::readHeader()
{
    uint16 magic;

    AGReadMAGIC(mReader, &magic);
    AGReadMAJORVERSION(&mReader, &mMajorVersion);
    AGReadMINORVERSION(&mReader, &mMinorVersion);

    if (magic != ((AG_PROTOCOL_MAGIC_HIGH << 8) | (AG_PROTOCOL_MAGIC_LOW)))

```

```

ACE_ERROR_RETURN ((LM_ERROR,
                  "%n: ERROR wrong magic %d != %d\n",
                  magic,
                  ((AG_PROTOCOL_MAGIC_HIGH << 8) | (AG_PROTOCOL_MAGIC_LOW))),
                  -1);

if (mMajorVersion != AG_PROTOCOL_MAJOR_VERSION)
    ACE_ERROR_RETURN ((LM_ERROR,
                      "%n: ERROR wrong major version client talking %d != server talking %d\n",
                      mMajorVersion,
                      AG_PROTOCOL_MAJOR_VERSION),
                      -1);

if (mMinorVersion != AG_PROTOCOL_MINOR_VERSION)
    ACE_ERROR_RETURN ((LM_ERROR,
                      "%n: ERROR wrong minor version client talking %d != server talking %d\n",
                      mMinorVersion,
                      AG_PROTOCOL_MINOR_VERSION));

return 0;

int
MALSession::readCommands()
{
    int err = 0;
    int count = 0;
    bool done = false;

    do {
        int32 command = AGReadCompactInt(&mReader);
        int32 len = AGReadCompactInt(&mReader);

        switch(command) {
            case AG_END_CMD:
                AGReadEND(&mReader);
                done = TRUE;
                break;
            case AG_EXPANSION_CMD:
                err = readExpansionCmd(len);
                break;
            case AG_HELLO_CMD:
                err = readHelloCmd();
                break;
            case AG_DEVICEINFO_CMD:
                err = readDeviceInfoCmd();
                break;
            case AG_OPENDATABASE_CMD:
                err = readOpenDatabaseCmd();
                break;
            case AG_CLOSEDATABASE_CMD:
                err = readCloseDatabaseCmd();
                break;
            case AG_RECORD_CMD:
                err = readRecordCmd();
                break;
            case AG_UNKNOWNDATABASE_CMD:
                err = readUnknownDatabaseCmd();
                break;
            case AG_NEWIDS_CMD:
                err = readNewIdsCmd();
                break;
            case AG_PING_CMD:
                err = readPingCmd();
                break;
            case AG_XMLDATA_CMD:
                err = readXMLDataCmd();
                break;
            default:
                ACE_ERROR_RETURN ((LM_ERROR,
                                  "%n: ERROR unknown command %d\n",
                                  command),
                                  -1);
                break;
        }
    } while (!done);
}

```

```

}

{
    AGWriteCOODBYE(&mWriter, AG_DONE_STATUS, 1, "Error Processing Request");
    AGWriteEND(&mWriter);

    ACE_ERROR_RETURN ((LM_ERROR,
                      "%n: ERROR handling command %s\n",
                      AGProtocolCommandName((AGCommand)command)),
                      -1);

    count++;
    while (!done);
    return 0;

int
MALSession::sendResponse()
{
    // write header
    AGWriteMAGIC(&mWriter);
    AGWriteMAJORVERSION(&mWriter, AG_PROTOCOL_MAJOR_VERSION);
    AGWriteMINORVERSION(&mWriter, AG_PROTOCOL_MINOR_VERSION);
    MALWriter writer(&mWriter);

    if (mIsInitialSession)
        mDispatcherP->sendInitialResponses(&writer, &mClientInfo);

    // clean this up
    time_t now = ACE_OS::time();
    char timeStr[64];
    (void) ACE_OS::ctime_r(&now, timeStr, sizeof(timeStr));

    AGWriteCOOKIE(&mWriter, ACE_OS::strlen(timeStr) + 1, timeStr);
    AGWriteSENDERDEVICEINFO(&mWriter, true);
    AGWriteCOODBYE(&mWriter, AG_CALLAGAIN_STATUS, 0, NULL);
}
else
{
    mDispatcherP->sendNormalResponses(&writer, &mClientInfo);
    AGWriteCOODBYE(&mWriter, AG_DONE_STATUS, 0, NULL);
}

AGWriteEND(&mWriter);
return 0;

int
MALSession::readExpansionCmd(int len)
{
    int32 command;
    int32 commandlen;
    void *buf = NULL;

    AGReadEXPANSION(&mReader, &command, &commandlen, &buf);
    if (!buf)
        free(buf);

    return 0;

int
MALSession::readHelloCmd()
{
    char *username;
    void *userCookie;
    uint8 digestAuth[16], nonce[16];
    int32 availableBytes, cookieLength;
    uint32 uid;
}

```

```

        if (mConduitP != NULL)
        {
            mConduitP->closeDatabase();
            mConduitP = NULL;
            return 0;
        }

        int
        MALSession::readRecordCmd()
        {
            int32 uid;
            ARecordStatus mod;
            int32 recordDataLength;
            void *recordData;
            int32 platformDataLength;
            void *platformData;

            AReadRECORD(&mReader, &uid, &mod,
                        &recordDataLength, &recordData,
                        &platformDataLength, &platformData);

            if (mConduitP != NULL)
            {
                mConduitP->record(uid, mod,
                                recordDataLength, recordData,
                                platformDataLength, platformData);
            }

            free(recordData);
            free(platformData);
            return 0;
        }

        int
        MALSession::readUnknownDatabaseCmd()
        {
            char *dbname;

            AReadUNKNOWNDATABASE(&mReader, &dbname);

            MALConduit *conduitP = mDispatcherP->getConduitForDatabase(dbname);
            if (conduitP != NULL)
            {
                conduitP->unknownDatabase(dbname);
                free(dbname);
                return 0;
            }

            int
            MALSession::readNewIdsCmd()
            {
                AGArray *newids = NULL;
                int32 count;

                AReadNEWIDS(&mReader, &newids);
                if (newids != NULL)
                {
                    count = AGArrayCount(newids);
                    if (count < 1 || count > 2 > 0)
                    {
                        AGArrayFree(newids);
                        return -1;
                    }
                    else if (mConduitP != NULL)
                    {
                        MALIds ids(newids);
                        mConduitP->newIds(&ids);
                        AGArrayFree(newids);
                    }
                    return 0;
                }
            }

            int
            MALSession::readCloseDatabaseCmd()
        }
    }

    AGDigestSetOnNull(digestAuth);
    AGDigestSetOnNull(nonce);

    if (mMajorVersion == 0 && mMinorVersion == 0)
    {
        AReadHELLO(&mReader, &username, &digestAuth, &nonce, &availableBytes,
                    &cookieLength, &userCookie);
    }
    else
    {
        AReadHELLO2(&mReader, &username, &digestAuth, &nonce, &availableBytes,
                    &cookieLength, &userCookie, &uid);
    }

    // authenticate here

    mClientInfo.setUserName(username);
    mClientInfo.setAvailableBytes(availableBytes);

    if (cookieLength == 0)
    {
        mIsInitialSession = true;
    }
    else
    {
        mIsInitialSession = false;
        // read data out of cookie
        free(userCookie);
    }

    return 0;
}

int
MALSession::readDeviceInfoCmd()
{
    char *osName, *osVersion, *serialNumber, *language, *charset;
    int32 colorDepth, screenWidth, screenHeight, platformDataLength;
    void *platformData;

    AReadDEVICEINFO(&mReader, &osName, &osVersion,
                    &colorDepth, &screenWidth, &screenHeight,
                    &serialNumber, &language, &charset,
                    &platformDataLength, &platformData);

    mClientInfo.setOSName(osName);
    mClientInfo.setOSVersion(osVersion);
    mClientInfo.setSerialNumber(serialNumber);
    mClientInfo.setLanguage(language);
    mClientInfo.setCharset(charset);

    free(platformData);

    return 0;
}

int
MALSession::readOpenDatabaseCmd()
{
    char *dbname;

    AReadOPENDATABASE(&mReader, &dbname);

    mConduitP = mDispatcherP->getConduitForDatabase(dbname);
    if (mConduitP != NULL)
    {
        mConduitP->openDatabase(dbname);
        free(dbname);
        return 0;
    }

    int
    MALSession::readCloseDatabaseCmd()
    {

```



```

int
MALSession::readPingCmd()
{
    AGRadping(smReader);
    return 0;
}

int
MALSession::readXMLDataCmd()
{
    int32 xmlLen = 0;
    void *xmlData = NULL;
    AGRadXMLData(smReader, &xmlLen, &xmlData);
    if (xmlData != NULL)
    {
        free(xmlData);
    }
    return 0;
}

// BLClientInfoManager.cpp
#include "BLClientInfoManager.h"
#include "ace/OS.h"

BLClientInfoManager* BLClientInfoManager::sInstanceP = NULL;

BLClientInfoManager*
BLClientInfoManager::instance()
{
    if (sInstanceP == NULL)
        sInstanceP = new BLClientInfoManager();
    return sInstanceP;
}

BLClientInfoManager::BLClientInfoManager()
: mDefaultInfoP(NULL)
{
}

BLClientInfoManager::~BLClientInfoManager()
{
}

void
BLClientInfoManager::initialize()
{
    BLClientInfo *infoP = new BLClientInfo();
    infoP->setLanguage("en-us");
    infoP->setCharset("ISO-8859-1");
    infoP->setOSName("Palm OS");
    infoP->setOSVersion("3.0");
    infoP->setScreenWidth(160);
    infoP->setScreenHeight(160);
    infoP->setBitDepth(2);
    infoP->setAcceptType(BLTransformManager::kPalmBMPTType);
    infoP->setUserAgent("UPGI UP/4.0 (compatible; BLServer 1.0)");
    infoP->setAcceptString("text/x-wap.wml, image/vnd.wap.wbmp, */*");
    infoP->setAcceptString("*/*");

    InfoPair pair;
    pair.first = "Blazer";
    pair.second = infoP;
    mInfoVector.push_back(pair);
    mInfoVector.push_back(pair);
}

BLClientInfo::BLClientInfo()
{
    infoP->setLanguage("en-us");
    infoP->setCharset("ISO-8859-1");
    infoP->setOSName("Palm OS");
    infoP->setOSVersion("3.0");
    infoP->setScreenWidth(160);
    infoP->setScreenHeight(160);
    infoP->setBitDepth(2);
    infoP->setUserAgent("UPGI UP/4.0 (compatible; BLServer 1.0)");
    infoP->setAcceptString("text/x-wap.wml, image/vnd.wap.wbmp, */*");
    infoP->setAcceptString("*/*");
    mDefaultInfoP = infoP;
}

void
BLClientInfoManager::finalize()
{
    // cleanup
}

BLClientInfo*
BLClientInfoManager::getUserAgentInfo(const char *uaString)
{
    if (uaString != NULL)
    {
        for (std::vector<InfoPair>::iterator i = mInfoVector.begin();
             i != mInfoVector.end();
             i++)
        {
            const char *cp = i->first;
            const char *up = uaString;
            while (true)
            {
                if (*cp == '\\0') // match
                    return i->second;
                if (*up == '\\0')
                    break;
                if (*cp != *up)
                    break;
                cp++;
                up++;
            }
        }
        return mDefaultInfoP;
    }
    // BLLog.cpp
    #include "BLLog.h"
    #include "ace/Log_Mag.h"
    BLAccessLog* BLAccessLog::sInstanceP = NULL;
    BLErrorLog* BLErrorLog::sInstanceP = NULL;
    int
    BLErrorLog::initialize()
    {
}

```

```

ACE_NEW_RETURN(sInstanceP,
               BLHttpRequest(),
               -1);

if (!sInstanceP->mFile.is_open())
{
    delete sInstanceP;
    sInstanceP = NULL;
    ACE_ERROR_RETURN((LM_ERROR, "%p: Could not open error log for writing."), -1);
}

return 0;
}

int
BLAccessLog::initialize()
{
    ACE_NEW_RETURN(sInstanceP,
                   BLAccessLog(),
                   -1);

    if (!sInstanceP->mFile.is_open())
    {
        delete sInstanceP;
        sInstanceP = NULL;
        ACE_ERROR_RETURN((LM_ERROR, "%p: Could not open access log for writing."), -1);
    }

    return 0;
}

void
BLHttpRequest::finalize()
{
    delete sInstanceP;
    sInstanceP = NULL;
}

void
BLAccessLog::finalize()
{
    delete sInstanceP;
    sInstanceP = NULL;
}

BLHttpRequest::BLHttpRequest()
: BLLog("error_log")
{
}

BLAccessLog::BLAccessLog()
: BLLog("access_log")
{
}

BLLog::BLLog(const char* path)
: mFile(path, std::ios::app)
{
}

// BLHttpRequest.cpp

#define ACE_BUILD_SVC_DLL
#include "BLHttpRequest.h"
#include "MALConduitManager.h"
#include "BLTransportManager.h"
#include "BLClientInfoManager.h"
#include "MALConfig.h"
#include "MALConstants.h"
#include "BLLog.h"

#include "ace/Get_Opt.h"

```

```

int
BLHttpRequest::init (int argc, char *argv[])
{
    int err;
    // parse arguments
    if (parseArgs(argc, argv) != 0)
        return -1;

    // initialize MAL stuff
    MALConfig config;
    if (config.readConfigFile(mConfigFilename) < 0)
        return -1;

    if (BLAccessLog::initialize() < 0 ||
        BLHttpRequest::initialize() < 0)
        return -1;

    MALConduitManager::instance()->initialize();

    // and proxy stuff
    BLTransportManager::instance()->initialize();
    if (BLTransportManager::instance()->initialize() < 0)
    {
        return -1;
    }
    BLClientInfoManager::instance()->initialize();

    // initialize acceptor
    service_port_ = config.getPort();
    if (config.getInterface() != NULL)
        err = service_addr_.set (config.getPort(), config.getInterface());
    else
        err = service_addr_.set (config.getPort());

    if (err < 0)
        return -1;

    if (open (service_addr_,
              ACE_Reactor::instance (),
              0, 0, 0,
              msScheduleStrategy,
              "MAL Server",
              "MAL Service") == -1)
        ACE_ERROR_RETURN ((LM_ERROR,
                           "\n: %p on port %d\n",
                           "acceptor:open failed",
                           service_addr_.get_port_number (),
                           -1));

    BLHttpRequest::write("Server Started");

    return 0;
}

// This method is automatically called when the server is shutdown.

int
BLHttpRequest::fini (void)
{
    // Cleanup
    BLHttpRequest::write("Server Shutting Down");

    MALConduitManager::instance()->finalize();
    BLTransportManager::instance()->finalize();
    BLTransportManager::instance()->finalize();
    BLClientInfoManager::instance()->finalize();
    BLAccessLog::finalize();
    BLHttpRequest::finalize();

    return 0;
}

int

```

```

BLHttpAcceptor::parseArgs (int argc, char *argv[])
{
    this->ConfigFilename = KWALDefaultFilename;
    ACE_Get_Opt get_opt (argc, argv, "f:", 0);
    for (int c; (c = get_opt ()) != -1; )
    {
        switch (c)
        {
            case 'f':
                this->mConfigFilename = get_opt.optarg;
                break;
            default:
                ACE_ERROR_RETURN ((LM_ERROR,
                    "bn:\n[-f config-file]\n%a", 1),
                    -1);
        }
    }
    return 0;
}

BLHttpAcceptor::make_svc_handler (BLHandler *handler)
{
    ACE_NEW_RETURN (handler,
        BLHttpHandler,
        -1);
    return 0;
}

// The following is a "Factory" used by the ACE_Service Config and
// svc.conf file to dynamically initialize the state of the server.

ACE_SVC_FACTORY_DEFINE (BLHttpAcceptor)
// BLHttpBase.cpp

#include "BLHttpBase.h"
#include "BLTransportStream.h"
#include "BLUtils.h"

#define kLineBufSize 2048
#define kBufSize 2048

BLHttpBase::BLHttpBase()
: mContentType(BLTransformManager::kUnknownType),
  mContentLength(-1),
  mMajorVersion(0),
  mMinorVersion(0),
  mChunkedEncoding(false)
{
}

void
BLHttpBase::init(ACE_SOCK_Stream *streamP, ACE_Time_Value *timeoutP)
{
    mStreamP = streamP;
    timeval t = timeoutP->operator timeval();
    mTimeout.set(t);
}

int
BLHttpBase::readMessage(BLTransportStream *transportP)
{
    int err;

    // check for chunked-encoding first since some broken
    // servers set content length and chunked-encoding
    if (mChunkedEncoding)
    {
        transportP->init(BLTransportStream::kUnknownLength, mContentType);
        err = readChunkedEncoding(transportP);
    }
}

```

```

    else if (mContentLength > 0)
    {
        transportP->init(mContentLength, mContentType);
        err = readChunk(mContentLength, transportP);
    }
    else
    {
        transportP->init(BLTransportStream::kUnknownLength, mContentType);
        err = readFull(transportP);
    }

    char c = 0;
    transportP->put(&c, 0, true);
    return err;
}

int
BLHttpBase::readChunk(int length, BLTransportStream *transportP)
{
    char buf[kBufSize];
    while (true)
    {
        int numRead = mStreamP->recv(&buf, BLUtils::min(length, kBufSize), &mTimeout);
        if (numRead <= 0)
            return -1;
        if ((transportP->put(buf, numRead, false) < 0)
            return -1;
        length -= numRead;
        if (length == 0)
            break;
    }
    return 0;
}

int
BLHttpBase::readChunkedEncoding(BLTransportStream *transportP)
{
    char buf[kLineBufSize];
    while (true)
    {
        if (readline(buf, kLineBufSize) < 0)
            return -1;
        int chunkSize = ACE_OS::strtol(buf, NULL, 16);
        if (chunkSize < 0)
            return -1;
        if (chunkSize == 0)
            break;
        if (readChunk(chunkSize, transportP) < 0)
            return -1;
        if (readline(buf, kLineBufSize) < 0)
            return -1;
    }

    // skip entity headers (if any)
    do {
        if (readline(buf, kBufSize) < 0)
            return -1;
    } while (buf[0] != '\0');
    return 0;
}

int
BLHttpBase::readFull(BLTransportStream *transportP)
{
    char buf[kBufSize];
    int numRead;
    while (true)
    {

```

```

numRead = mStreamP->recv(buf, kBufSize, &mTimeout);
if (numRead < 0)
    return -1;
if (numRead == 0)
    break;

if (transportP->put(buf, numRead, false) < 0)
    return -1;

return 0;

int
BLHttpBase::sendHeader(const char* header, const char* value)
{
    char buf[kBufSize];
    ACE_OS::sprintf(buf, "%s: %s\r\n", header, value);
    return mStreamP->send_n(buf, ACE_OS::strlen(buf), &mTimeout);
}

int
BLHttpBase::sendHeader(const char* header, int value)
{
    char buf[kBufSize];
    ACE_OS::sprintf(buf, "%s: %d\r\n", header, value);
    return mStreamP->send_n(buf, ACE_OS::strlen(buf), &mTimeout);
}

int
BLHttpBase::sendHeaders(BLHttpRequestInfo* infoP)
{
    if (infoP == NULL)
        return 0;

    BLHttpRequestInfo::HeaderVector* headers = infoP->getHeaders();
    for (BLHttpRequestInfo::HeaderVector::iterator i = headers->begin();
         i != headers->end();
         ++i)
    {
        if (sendHeader(i->first, i->second) < 0)
            return -1;
        return 0;
    }

    int
    BLHttpBase::parseHeaders()
    {
        char buf[kLineBufSize];
        while (true)
        {
            if (readline(buf, kLineBufSize) == -1)
                return -1;
            if (buf[0] == '\0')
                break;
            char* cp = ACE_OS::strchr(buf, ':');
            if (cp == NULL)
                return -1;
            *cp = '\0';
            while (*(++cp) == ' '); // skip spaces
            if (ACE_OS::strcasecmp(buf, "content-type") == 0)
            {
                mContentType = BLTransformManager::instance()->stringToContentType(cp);
            }
            else if (ACE_OS::strcasecmp(buf, "content-length") == 0)
            {

```

```

        mContentLength = ACE_OS::atoi(cp);
    }
    else if (ACE_OS::strcasecmp(buf, "transfer-encoding") == 0)
    {
        if (ACE_OS::strncmp(cp, "chunked", sizeof("chunked") - 1) == 0)
            mChunkedEncoding = true;
    }
    else if (handleHeader(buf, cp) < 0)
        return -1;
    return 0;

int
BLHttpBase::readline(char* buf, int size)
{
    char c = 0;
    int rc = 0;
    int i;
    for (i = 0; i < size - 1; i++)
    {
        if ((rc = mStreamP->recv_n(&c, 1, 0, &mTimeout)) == 1)
        {
            if (c == '\n')
                break;
            if (c == '\r') {
                i--;
                continue;
            }
            buf[i] = c;
        }
        else
            return -1;
    }
    buf[i] = '\0';
    return 0;

int
BLHttpBase::sendString(const char* str)
{
    int len = ACE_OS::strlen(str);
    if (mStreamP->send_n(str, len) != len)
        return -1;
    return 0;

// BLHttpHandler.cpp
#include "BLHttpHandler.h"
#include "BLProxySession.h"
#include "MPLSession.h"
#include "BLLog.h"

#define kLineBufSize 512
#define kHTTPBufSize 2048
#define kMaxRequestPerSession 50
#define kHTTPTimeout 3 * 60

BLHttpHandler::BLHttpHandler(ACE_Thread_Manager*)
: mUserAgent(NULL),
  mType(kGetType)
{
}

BLHttpHandler::~BLHttpHandler()
{
    delete[] mUserAgent;
}

int

```

```

BLHttpHandler::open (void *)
{
    // Shut off non-blocking IO if it was enabled...
    if (this->peer().disable (ACE_NONBLOCK) == -1)
        ACE_ERROR_RETURN ((LM_ERROR,
            "sp\n",
            "disable"),
            -1);

    // Spawn a new thread of control to handle the
    // client using a thread-per-connection concurrency model. Note
    // that this implicitly uses the <ACE_Thread_Manager::instance> to
    // control all the threads.
    if (this->activate (THR_BOUND | THR_DETACHED) == -1)
        ACE_ERROR_RETURN ((LM_ERROR,
            "sp\n",
            "spawn"),
            -1);

    return 0;
}

int
BLHttpHandler::svc()
{
    char pathBuf[kHTTPBufSize];
    int numRequests = kMaxRequestsPerSession;

    ACE_Time_Value timeout(kHttpTimeout);
    BLHttpBase::init(speer(), &timeout);

    do {
        mRequestInfo.clear();

        if (readRequestLine(pathBuf, kHTTPBufSize) < 0)
            return -1;

        if (mMajorVersion < 1 || (mMajorVersion == 1 && mMinorVersion < 1))
            numRequests = 1;

        if (mType == kPostType &&
            mContentLength > 0 &&
            mContentType != BLTransformManager::kUnknownType)
        {
            mRequestInfo.setPostData(this);
        }

        if (parseHeaders() < 0)
            return -1;

        if (mContentType != BLTransformManager::kUnknownType)
            mRequestInfo.setHeader("Content-Type",
                BLTransformManager::instance()->contentTypeOfString(mContentType));

        BLAccessLog::write(mUserAgent, mStreamP, pathBuf);

        int err;
        if (ACE_OS::strcmp(pathBuf, "/sync") == 0)
        {
            if (mType != kPostType)
                return -1;
            MALSession malSession;
            err = malSession.doSession(mStreamP);
        }
        else
        {
            BLProxySession proxySession;
            err = proxySession.doSession(pathBuf, this, mStreamP);
        }
        if (err < 0)
            return -1;
    } while (--numRequests > 0);
}

```

```

    return 0;
}

int
BLHttpHandler::readRequestLine(char *pathBuffer,
    int bufferSize)
{
    char buf(kHTTPBufSize);

    do {
        if (readline(buf, kHTTPBufSize) < 0)
            return -1;
        while (buf[0] == '\0');

        char *cp = buf;

        // Request Type
        if (ACE_OS::strcmp(buf, "POST", sizeof("POST") - 1) == 0)
        {
            mType = kPostType;
            cp += sizeof("POST") - 1;
        }
        else if (ACE_OS::strcmp(buf, "GET", sizeof("GET") - 1) == 0)
        {
            mType = kGettype;
            cp += sizeof("GET") - 1;
        }
        else
            return -1;

        // skip whitespace
        while (*cp == ' ' || *cp == '\t')
            if (*cp == '\0')
                return -1;
            cp++;

        // Path
        int i = 0;
        while (*cp != ' ' && *cp != '\t' && i < bufferSize - 1)
        {
            if (*cp == '\0')
                return -1;
            pathBuffer[i++] = *cp++;

            pathBuffer[i] = '\0';

            // skip whitespace
            while (*cp == ' ' || *cp == '\t')
            {
                if (*cp == '\0')
                    return -1;
                cp++;
            }

            // HTTP
            if (ACE_OS::strcmp(cp, "HTTP/", sizeof("HTTP/") - 1) != 0)
                return -1;
            cp += sizeof("HTTP/") - 1;

            // version
            mMajorVersion = ACE_OS::strtol(cp, &cp, 10);
            if (cp == NULL || *cp != '.')
                return -1;
            mMinorVersion = ACE_OS::strtol(cp + 1, &cp, 10);
            if (cp == NULL)
                return -1;

            return 0;
        }

        int
        BLHttpHandler::handleHeader(const char* name, const char* value)
    }

```

```

if (ACE_OS::strcasecmp(name, "user-agent") == 0)
{
    if (mUserAgent != NULL)
        delete mUserAgent;

    mUserAgent = BUUtil::strup(value);
}
else if (ACE_OS::strcasecmp(name, "content-encoding") == 0 ||
         ACE_OS::strcasecmp(name, "cookie") == 0)
{
    mRequestInfo.setHeader(name, value);
}

return 0;
}

int
BLHttpHandler::feed(BLTransportStream *transportP)
{
    if (mContentLength == -1)
    {
        transportP->init(mContentLength, mContentType);
        return 0;
    }

    return readMessage(transportP);
}

// BLHttpRequest.cpp
#include "BLHttpRequest.h"
#include "BLTransportManager.h"
#include "BLClientInfo.h"
#include "ace/Auto_Ptr.h"

#ifdef ACE_WIN32
#include <strstream>
#else
#include <sstream>
#endif // ACE_WIN32

#define kLineBufSize 2048
#define kBufSize 2048
#define kMaxRedirects 10

// don't call HTTPBase constructor because we do everything in init
BLHttpRequest::BLHttpRequest()
{
}

void
BLHttpRequest::init(ACE_SOCK_Stream *streamP)
{
    mContentType = BLTransportManager::kUnknownType;
    mMajorVersion = 0;
    mMinorVersion = 0;
    mProxyInfoP = NULL;
    mLocation = NULL;
    mChunkedEncoding = false;
    mContentLength = -1;

    ACE_Time_Value timeout(ACE_DEFAULT_TIMEOUT);
    BLHttpBase::init(streamP, timeout);
}

BLHttpRequest::~BLHttpRequest()
{
    if (mLocation != NULL)
        delete[] mLocation;

    if (mStreamP != NULL)
    {
        mStreamP->close();
        delete mStreamP;
    }
}

```

```

int
BLHttpRequest::readMessage()
{
    if (mStreamP == NULL)
        return -1;

    char buf(kLineBufSize);
    const char *hostname = urlP->getHostname();
    const char *path = urlP->getPath();
    mProxyInfoP = infoP;
    BLTransportFeeder *postP = infoP == NULL ? NULL : infoP->getPostData();

    int commandsSize =
        ACE_OS::strlen(path)
        + 20 // Extra
        + 1 // NUL byte
        + 16; // Protocol filler...

    if (commandsSize > kLineBufSize)
        return -1;

    ACE_OS::sprintf (buf,
        "%s %s HTTP/1.1\r\n",
        postP == NULL ? "GET" : "POST",
        path);

    // send headers
    if (sendString(buf) < 0 ||
        sendHeader("Accept-Charset", clientInfoP->getCharset()) < 0 ||
        sendHeader("Accept-Language", clientInfoP->getLanguage()) < 0 ||
        sendHeader("Accept", clientInfoP->getAcceptString()) < 0 ||
        sendHeader("User-Agent", clientInfoP->getUserAgent()) < 0 ||
        sendHeader("Host", hostname) < 0)
        return -1;

    if (infoP != NULL)
    {
        if (sendHeaders(infoP) < 0)
            return -1;
    }

    if (postP == NULL)
    {
        if (sendString("\r\n") < 0)
            return -1;
    }
    else
    {
        if (postP->feed(this) < 0)
            return -1;
    }

    int status = readStatusLine();
    if (parseHeaders() == -1)
        return -1;
    else
        return status;
}

int
BLHttpRequest::readFile(BLTransportStream *transportP)
{
    int err = readMessage(transportP);

    mStreamP->close();
    delete mStreamP;
    mStreamP = NULL;

    return err;
}

int
BLHttpRequest::readStatusLine()

```

```

    char linebuff[kLineBufSize];
    char tokbuff[kLineBufSize];

    int status = 0;

    if (readline(linebuff, kLineBufSize) == -1)
        return -1;
    istrstream is(linebuff);

    if (!is.getline(tokbuff, kLineBufSize, '/') ||
        !((ACE_OS::strcmp(tokbuff, "HTTP") == 0) ||
          !((is >> mMajorVersion) ||
            !((is.get() == '.') ||
              !((is >> mMinorVersion) ||
                !((is >> status) ||
                  !((is.getline(tokbuff, kLineBufSize))
                    return -1;

    return status;
}

int
BLHttpRequest::handleHeader(const char* name, const char* value)
{
    if (ACE_OS::strcmp(name, "location") == 0)
    {
        if (mLocation != NULL)
            delete[] mLocation;
        mLocation = BLUtils::strdup(value);
    }
    else if (ACE_OS::strcmp(name, "content-encoding") == 0)
    {
        mRequestInfo.setHeader(name, value);
    }
    else if (ACE_OS::strcmp(name, "set-cookie") == 0)
    {
        mRequestInfo.setHeader(name, value);
    }
    if (mProxyInfo != NULL)
    {
        mProxyInfo->setHeader("Cookie", value);
    }
    return 0;
}

int
BLHttpRequest::init(int length, BLTransformManager::ContentType type)
{
    if (sendHeader("Content-Length", length) < 0 ||
        sendHeader("Content-Type",
                  BLTransformManager::instance()->contentTypeToString(type)) < 0 ||
        sendString("\r\n") < 0)
        return -1;

    return 0;
}

int
BLHttpRequest::put(char* data, int size, bool final)
{
    if (size > 0 && mStream->send_n(data, size, &timeout) != size)
        return -1;

    return 0;
}

void
BLHttpRequestFactory::initialize(BLTransportManager* managerP)
{
    managerP->registerProtocolHandler(BLUrl::kHttpProtocol, this);
}

void
BLHttpRequestFactory::finalize()
{
}

```

```

    BLHttpRequestFactory::getName()
    {
        return "http";
    }

    BLTransportRequest*
    BLHttpRequestFactory::requestFile(BLUrl* urlP, BLClientInfo* clientInfoP, BLHttpRequestInfo*
    infoP)
    {
        int numRedirects = 0;
        BLHttpRequest* requestP;
        ACE_NEW_RETURN(requestP,
            BLHttpRequest(),
            NULL);

        while (true)
        {
            ACE_SOCK_Stream* sockStreamP;
            ACE_NEW_RETURN(sockStreamP,
                ACE_SOCK_Stream(),
                NULL);

            requestP->init(sockStreamP);
            if (mConnector.connect(*sockStreamP, *urlP->getAddr()) == -1)
            {
                delete requestP;
                return NULL;
            }

            if (requestP->requestFile(urlP, clientInfoP, infoP) < 0)
            {
                delete requestP;
                return NULL;
            }
            if (requestP->getLocation() == NULL)
            {
                if (numRedirects > 0)
                    requestP->getRequestInfo()->setHeader("Content-Location", urlP->toString());
                return requestP;
            }
            if (numRedirects >= kMaxRedirects)
            {
                delete requestP;
                return NULL;
            }
            // eat up the page
            BLNullTransportStream nullStream;
            requestP->readFile(&nullStream);

            if (BLUrl::isRelative(requestP->getLocation()))
            {
                if (urlP->setRelative(urlP, requestP->getLocation()) < 0)
                {
                    delete requestP;
                    return NULL;
                }
            }
            else
            {
                if (urlP->set(requestP->getLocation()) < 0)
                {
                    delete requestP;
                    return NULL;
                }
            }

            numRedirects++;
        }
        // BLProxySession.cpp
    }
}

```

```
#include "BLProxySession.h"
#include "BLClientInfoManager.h"
#include "BLTransportStream.h"
#include "BLTransportManager.h"
#include "BLTransformation.h"
#include "BLHttpBase.h"
#include "ace/Auto_Ptr.h"

#define kMaxProtocolLength 10
#define kMaxUrlLength 1024
#define kErrorBufSize 2048
```

```
class ProxyStream : public BLTransportStream, public BLHttpBase
```

```
{
public:
    ProxyStream(ACE_SOCK_Stream *ios, BLHttpRequestInfo *infoP, bool chunked)
        : mWriteHeader(false),
          mType(BLTransportManager::kUnknownType),
          mInfoP(infoP),
          mChunked(chunked),
          mBad(true)
    {
        ACE_Time_Value t(ACE_DEFAULT_TIMEOUT);
        BLHttpBase::init(ios, t);
    }

    virtual int init(int length, BLTransportManager::ContentType type);
    virtual int put(char *data, int size, bool final);
protected:
```

```
int error()
{
    mBad = true;
    return -1;
}
```

```
BLTransportManager::ContentType mType;
BLHttpRequestInfo *mInfoP;
bool mWriteHeader;
bool mChunked;
bool mBad;
int mLength;
};
```

```
int
ProxyStream::init(int length, BLTransportManager::ContentType type)
{
    mLength = length;
    mType = type;
    mBad = false;

    if (sendString("HTTP/1.1 200 OK\r\n") < 0 ||
        sendHeader("Content-Type",
                    BLTransportManager::instance()->contentTypeToString(mType)) < 0 ||
        sendHeaders(mInfoP) < 0)
        return error();
}
```

```
if (mLength != BLTransportStream::kUnknownLength)
{
    if (sendHeader("Content-Length", length) < 0)
        return error();

    mChunked = false;
}
else if (mChunked)
{
    if (sendHeader("Transfer-Encoding", "chunked") < 0)
        return error();
}

if (sendString("\r\n") < 0)
    return error();
```

```
return 0;
}

int
ProxyStream::put(char *data, int size, bool final)
```

```
{
    if (mBad)
        return -1;

    if (mChunked) // chunked
    {
        if (size > 0)
        {
            char buf[32];
            ACE_OS::sprintf(buf, "%x\r\n", size);

            if (sendString(buf) < 0 ||
                mStream->send_n(data, size) != size ||
                sendString("\r\n") < 0)
                return error();
        }
    }
    if (final)
    {
        if (sendString("0\r\n\r\n") < 0)
            return error();
    }
    else // not chunked
    {
        if (size > 0)
        {
            if (mStream->send_n(data, size) != size)
                return error();
        }
    }

    return 0;
}
```

```
BLProxySession::BLProxySession()
{
}
```

```
int
BLProxySession::doSession(char *path, BLHttpHandler *handlerP, ACE_SOCK_Stream *ios)
{
    BLClientInfo *clientInfoP =
        BLClientInfoManager::instance()->getUserAgentInfo(handlerP->getUserAgent());
    if (clientInfoP == NULL)
        return -1;

    BLUrl url;
    if (parsePath(path, &url) < 0)
        return error("Operation not permitted.", handlerP, clientInfoP, ios);

    BLTransportRequest *requestP = BLTransportManager::instance()->requestFile(&url,
        clientInfoP,
```

```
handlerP->getRequestInfo());
    auto_ptr<BLTransportRequest> requestPtr(requestP);
    if (requestP == NULL)
        return error("Unable to connect to server.", handlerP, clientInfoP, ios);

    ProxyStream *outputP = getProxyStream(handlerP, ios, requestP->getRequestInfo());
    auto_ptr<BLTransportStream> outputPtr(outputP);
    if (outputP == NULL)
        return -1;

    BLTransformation *transformP =
        BLTransportManager::instance()->getTransformation(clientInfoP,
        requestP->getContentType(),
        outputP);
```



```

    if (transform == NULL)
        return error("Unsupported content type.", handlerP, clientInfoP, ios);
    auto_ptr<BLTransformation> transformPtr(transformP);
    if (requestP->readFile(transformP->getTransportStream()) < 0)
        return -1;
    return 0;
}

ProxyStream*
BLProxySession::getProxyStream(BLHttpHandler *handlerP, ACE_SOCK_Stream *ios,
    BLHttpRequestInfo *infoP)
{
    bool chunked = handlerP->getMajorVersion() >= 1 && handlerP->getMinorVersion() >= 1;
    return new ProxyStream(ios, infoP, chunked);
}

int
BLProxySession::parsePath(char *path, BUUrl *urlP)
{
    if (BUUrl::isAbsolute(path))
    {
        if (urlP->set(path) < 0)
            return -1;
    }
    else
    {
        if (path[0] != '/')
            return -1;
        path++;
        // protocol
        char *cp = ACE_OS::strchr(path, '/');
        if (cp == NULL)
            return -1;
        *cp = '\0';
        BUUrl::ProtocolType protocol = BUUrl::stringToProtocolType(path);
        // port
        char *port = cp + 1;
        cp = ACE_OS::strchr(port, '/');
        if (cp == NULL)
            return -1;
        *cp = '\0';
        int portNum = ACE_OS::atoi(port);
        if (portNum <= 0)
            return -1;
        char *urlString = cp + 1;
        if (urlP->set(protocol, urlString))
            return -1;
        urlP->setPort(portNum);
    }
    return 0;
}

int
BLProxySession::error(char *error, BLHttpHandler *handlerP, BLClientInfo *clientInfoP,
    ACE_SOCK_Stream *ios)
{
    char buf[kErrorBufSize];
    ACE_OS::sprintf(buf, "<b>Server Error:</b><br>%s", error);
    ProxyStream *outputP = getProxyStream(handlerP, ios, NULL);
    auto_ptr<BLTransportStream> outputPtr(outputP);
    if (outputP == NULL)
        return -1;
    BLTransformation *transformP =
        BLTransformationManager::instance()->getTransformation(clientInfoP,
            BLTransformationManager::KHTMLType,
            outputP);
}

```

```

    if (transform == NULL)
        return -1;
    auto_ptr<BLTransformation> transformPtr(transformP);
    if (transformP->getTransportStream()->put(buf, ACE_OS::strlen(buf), true) < 0)
        return -1;
    return 0;
}

// BLTransportDefs.cpp
#include "BLHttpRequest.h"
#include "BLTransportManager.h"
BLHttpRequestFactory httpRequestFactory;
BLTransportRequestFactory* BLTransportManager::sFactoryDefs[] =
{
    sHttpRequestFactory,
    NULL
};
// BLTransportManager.cpp
#include "BLTransportManager.h"
#include "ace/OS.h"
BLTransportManager* BLTransportManager::sInstanceP = NULL;
BLTransportManager*
BLTransportManager::instance()
{
    if (sInstanceP == NULL)
        sInstanceP = new BLTransportManager();
    return sInstanceP;
}
BLTransportManager::BLTransportManager()
{
    ACE_OS::memset(mHandlerTable, 0, BUUrl::KNumProtocols *
        sizeof(BLTransportRequestFactory*));
}
BLTransportManager::~BLTransportManager()
{
}
void
BLTransportManager::initialize()
{
    for (BLTransportRequestFactory** factoryP = sFactoryDefs;
        *factoryP != NULL;
        factoryP++)
        (*factoryP)->initialize(this);
}
void
BLTransportManager::finalize()
{
    for (BLTransportRequestFactory** factoryP = sFactoryDefs;
        *factoryP != NULL;
        factoryP++)
        (*factoryP)->finalize();
}
void
BLTransportManager::registerProtocolHandler(BUUrl::ProtocolType protocol,
    BLTransportRequestFactory *factoryP)
{
    if (protocol >= 0 && protocol < BUUrl::KNumProtocols)
        mHandlerTable[protocol] = factoryP;
}

```

```

)
ACE_NEW_RETURN(newBuf, char[1],
-1);
BLTransportRequestFactory*
BLTransportManager::getFactoryForProtocol(BLURL::ProtocolType protocol)
{
    if (protocol >= 0 && protocol < BLURL::kNumProtocols)
        return mHandlerTable[protocol];
    return NULL;
}

BLTransportRequest*
BLTransportManager::requestFile(BLURL *urlp, BLClientInfo *clientInfo, BLHttpReqInfo
*info)
{
    BLTransportRequestFactory *factoryP = getFactoryForProtocol(urlp->getProtocol());
    if (factoryP == NULL)
        return NULL;
    return factoryP->requestFile(urlp, clientInfoP, infoP);
}

// BLTransportStream.cpp
#include "BLTransportStream.h"
#include "ace/OS.h"
#define kDefaultBufSize 1024 * 8

BLAccumulatingStream::BLAccumulatingStream()
: mDataBuf(NULL),
mCurrentDataSize(0),
mTotalDataSize(-1)
{
}

BLAccumulatingStream::~BLAccumulatingStream()
{
    if (mDataBuf != NULL)
        delete[] mDataBuf;
}

int
BLAccumulatingStream::init(int length, BLTransformManager::ContentType type)
{
    if (length <= 0 || mDataBuf != NULL)
        return -1;

    mType = type;
    mTotalDataSize = length;
    ACE_NEW_RETURN(mDataBuf,
char[length],
-1);
    return 0;
}

int
BLAccumulatingStream::put(char *data, int size, bool final)
{
    if (mDataBuf == NULL)
    {
        mTotalDataSize = kDefaultBufSize;
        ACE_NEW_RETURN(mDataBuf,
char[kDefaultBufSize],
-1);
    }
    if (mCurrentDataSize + size > mTotalDataSize)
    {
        char *newBuf;
        int len;
        if (mCurrentDataSize + size < mTotalDataSize * 2)
            len = mTotalDataSize * 2;
        else
            len = mCurrentDataSize + size;
    }
}

```

```

ACE_NEW_RETURN(newBuf, char[1],
-1);
ACE_OS::memcpy(newBuf, mDataBuf, mCurrentDataSize);
delete[] mDataBuf;
mDataBuf = newBuf;
}

ACE_OS::memcpy(mDataBuf + mCurrentDataSize, data, size);
mCurrentDataSize += size;
if (final)
{
    int err = putFinal(mDataBuf, mCurrentDataSize);
    delete[] mDataBuf;
    mDataBuf = NULL;
    return err;
}
return 0;
}

// BLTransportWriter.cpp
#include "BLTransportWriter.h"
#include "BLTransportStream.h"

BLTransportStreamBuf::BLTransportStreamBuf(BLTransportStream *streamP)
: mStreamP(streamP)
{
    setp(mBuf, mBuf + kStreamBufSize - 1);
}

BLTransportStreamBuf::~BLTransportStreamBuf()
{
    if (mStreamP != NULL)
        mStreamP->put(mBuf, pptr() - mBuf, true);
}

int
BLTransportStreamBuf::overflow(int c)
{
    if (mStreamP == NULL)
        return EOF;
    if (c != EOF)
        *pptr() = c;
    mStreamP->put(mBuf, pptr() - mBuf + 1, false);
    setp(mBuf, mBuf + kStreamBufSize - 1);
    return c;
}

int
BLTransportStreamBuf::sync()
{
    if (mStreamP == NULL)
        return EOF;
    if (pptr() - mBuf > 0)
    {
        mStreamP->put(mBuf, pptr() - mBuf, false);
        setp(mBuf, mBuf + kStreamBufSize - 1);
    }
    return 0;
}

// BLURL.cpp
#include "BLURL.h"
#include "BLUtils.h"
#define kMaxPathLen 1024

```

```
BLUrl::ProtocolInfo BLUrl::sProtocolTable[kNumProtocols] = { ("http", 80) };
```

```
BLUrl::BLUrl()
: mProtocol(kUnknownProtocol),
  mHostName(NULL),
  mPath(NULL),
  mString(NULL)
{
}
```

```
BLUrl::~BLUrl()
{
    delete[] mHostName;
    delete[] mPath;
    delete[] mString;
}
```

```
void
BLUrl::clear()
```

```
{
    delete[] mHostName;
    delete[] mPath;
    delete[] mString;
    mHostName = NULL;
    mPath = NULL;
    mString = NULL;
}
```

```
int
BLUrl::set(ProtocolType protocol,
          char *hostname,
          char *path,
          int port)
```

```
{
    if (protocol < 0 || protocol >= kNumProtocols)
        return -1;
```

```
    // in case we are setting relative to self, we don't want to delete the old strings
    until we make copies
    char *newHostName;
    char *newPath;
    ACE_ALLOCATOR_RETURN(newHostName,
                          BUtills::strdup(hostname),
                          -1);
```

```
    ACE_ALLOCATOR_RETURN(newPath,
                          BUtills::strdup(path),
                          -1);

    clear();
    mHostName = newHostName;
    mPath = newPath;
    mProtocol = protocol;
    mAddr.set(port, mHostName);
    return 0;
}
```

```
int
BLUrl::set(ProtocolType protocol,
          char *hostname,
          char *path)
```

```
{
    if (protocol < 0 || protocol >= kNumProtocols)
        return -1;
```

```
    set(protocol, hostname, path, sProtocolTable[protocol].defaultPort);
    return 0;
}
```

```
int
BLUrl::set(ProtocolType protocol, char *urlString)
{
    if (protocol < 0 || protocol >= kNumProtocols)
        return -1;
    clear();
```

```
mProtocol = protocol;
```

```
char *cp = ACE_OS::strchr(urlString, '/');
if (cp != NULL)
```

```
{
    ACE_NEW_RETURN(mHostName,
                  char(cp - urlString + 1),
                  -1);
```

```
    ACE_OS::strcpy(mHostName, urlString, cp - urlString);
    mHostName[cp - urlString] = '\0';
```

```
    ACE_ALLOCATOR_RETURN(mPath,
                        BUtills::strdup(cp),
                        -1);
```

```
    else
```

```
{
    ACE_ALLOCATOR_RETURN(mHostName,
                        BUtills::strdup(urlString),
                        -1);
```

```
    ACE_ALLOCATOR_RETURN(mPath,
                        BUtills::strdup("/"),
                        -1);
```

```
    }
```

```
    int port;
```

```
    cp = ACE_OS::strchr(mHostName, ':');
```

```
    if (cp != NULL)
```

```
{
    *cp = '\0';
    cp++;
    port = ACE_OS::atoi(cp);
```

```
    }
```

```
    else
```

```
{
    port = sProtocolTable[protocol].defaultPort;
```

```
    mAddr.set(port, mHostName);
    return 0;
}
```

```
int
BLUrl::set(char *urlString)
```

```
{
    char buf[kMaxPathLen];
```

```
    char *cp = ACE_OS::strchr(urlString, '/');
    if (cp == NULL)
        return -1;
```

```
    int len = BUtills::min(kMaxPathLen, cp - urlString);
    ACE_OS::strcpy(buf, urlString, len);
    buf[len] = '\0';
    ProtocolType protocol = stringToProtocolType(buf);
```

```
    char *hostname = cp + 3;
    return set(protocol, hostname);
}
```

```
void
BLUrl::setPort(int port)
```

```
{
    mAddr.set_port_number(port);
}
```

```
int
BLUrl::setRelative(BLUrl *baseUrlP, char *path)
```

```
{
    if (path[0] == '/')
```

```
{
    set(baseUrlP->mProtocol, baseUrlP->mHostName, path,
        baseUrlP->mAddr.get_port_number());
    }
    else
```

```

{
    char buf[kMaxPathLen];
    const char *basePath = baseUrlP->mPath;
    const char *endP = basePath + ACE_OS::strlen(basePath) - 1;

    // back up to last '/'
    while (*endP != '/' && endP >= basePath)
        endP--;

    while (path[0] == '.' && path[1] == '.')
    {
        if (path[2] == '/')
            path += 3;
        else if (path[2] == '\0')
            path += 2;
        else
            return -1;
    }

    while (endP > basePath && * (--endP) != '/')
        ;

    int len = BUUtils::min(kMaxPathLen, endP - basePath + 1);
    ACE_OS::strncpy(buf, basePath, len);
    buf[len] = '\0';
    ACE_OS::strncat(buf, path, kMaxPathLen - len);

    set(baseUrlP->mProtocol, baseUrlP->mHostname, buf,
        baseUrlP->mAddr.get_port_number());

    return 0;
}

char*
BLUrl::toString()
{
    if (mProtocol == kUnknownProtocol)
        return NULL;
    if (mString != NULL)
        return mString;

    int len = ACE_OS::strlen(mProtocolTable[mProtocol].string)
        + 3 // len ://
        + ACE_OS::strlen(mHostname)
        + ACE_OS::strlen(mPath)
        + sizeof(":65536");

    ACE_NEW_RETURN(mString,
        char[len],
        NULL);

    if (mAddr.get_port_number() == mProtocolTable[mProtocol].defaultPort)
    {
        ACE_OS::sprintf(mString, "%s://%s%s", mProtocolTable[mProtocol].string,
            mHostname,
            mPath);
    }
    else
    {
        ACE_OS::sprintf(mString, "%s://%s:%d%s", mProtocolTable[mProtocol].string,
            mHostname, mAddr.get_port_number(),
            mPath);
    }

    return mString;
}

bool
BLUrl::isRelative(char *urlString)
{
    for (int i=0; i<kNumProtocols; i++)
    {
        char *cp = mProtocolTable[i].string;
        char *up = urlString;
        while (true)

```

```

{
    if (*cp == '\0') // match
        return ACE_OS::strncmp(up, ":", 3) != 0;
    if (*cp != '\0')
        break;
    if (*cp != *up)
        break;
    cp++;
    up++;
}

return true;
}

BLUrl::ProtocolType
BLUrl::stringToProtocolType(char *protocolString)
{
    for (int i=0; i<kNumProtocols; i++)
    {
        if (ACE_OS::strcasecmp(protocolString, mProtocolTable[i].string) == 0)
            return (ProtocolType)i;
    }

    return kUnknownProtocol;
}

#include "BLImageTransformer.h"
#include "BLClientInfo.h"

typedef unsigned short Word;
typedef unsigned char Byte;

#if (ACE_BYTE_ORDER == ACE_LITTLE_ENDIAN)
#define PALM_WORD(X) ((Word) (((X) >> 8) & 0xFF) | (((X) << 8) & 0xFF00))
#else
#define PALM_WORD(X) ((unsigned short) (X))
#endif

typedef struct {
    Word compressed; // Data format: 0=raw; 1=compressed
    Word hasColorTable; // if true, color table stored before bits[]
    Word reserved; // 14;
} BitmapFlagsType;

typedef struct {
    Word width;
    Word height;
    Word rowBytes;
    BitmapFlagsType flags;
    Byte pixelSize; // bits/pixel
    Byte version; // version of bitmap. This is vers 1
    Byte nextDepthOffset; // # of DWords to next BitmapType
    Word reserved; // from beginning of this one
    Word reserved2;
} BitmapType;

BLImageTransformer::BLImageTransformer(BLTransformManager::ContentType contentType,
    BLTransformManager::ContentType toType)
: BLTransformation(fromType, toType),
  mImageP(NULL),
  mOutputH(0),
  mOutputV(0),
  mOutputIsColor(0),
  mOutputQuality(75)
{
    GetImageInfo(mImageInfo);
}

int BLImageTransformer::init(BLTransportStream *outputTransportP,
    BLImageTransformerFactory *parentFactoryP)
{
    mOutputTransportP = outputTransportP;
    mParentFactoryP = parentFactoryP;
    return 0;
}

```

```

BLImageTransformer::BLImageTransformer()
{
    if (mImageP != NULL)
    {
        DestroyImage(mImageP);
    }
    // DestroyImageInfo(&mImageInfo);
}

BLTransportStream* BLImageTransformer::getTransportStream()
{
    return this;
}

void BLImageTransformer::setClientInfo(BLClientInfo *ciP)
{
    mClientInfoP = ciP;
}

int BLImageTransformer::readMem (const char *blobP, const size_t length)
{
    GetImageInfo(&mImageInfo);
    switch(mFromType)
    {
        case BLTransformManager::kJPEGType:
            strcpy(mImageInfo.filename, "image.jpg");
            break;
        case BLTransformManager::kGIFType:
            strcpy(mImageInfo.filename, "image.gif");
            break;
        case BLTransformManager::kPNGType:
            strcpy(mImageInfo.filename, "image.png");
            break;
        case BLTransformManager::kWBMPTType:
            strcpy(mImageInfo.filename, "image.wbmp");
            break;
    }

    //SetImageInfo(&mImageInfo, 0);
    mImageP = BlobToImage(&mImageInfo, blobP, length);
    if (mImageP == NULL)
    {
        //logError("Could not read image");
        return -1;
    }

    if (mImageP->next != NULL)
    {
        mImageP->next->previous = NULL;
        DestroyImages(mImageP->next);
        mImageP->next = NULL;
    }

    return 0;
}

int BLImageTransformer::putFinal(char *data, int size)
{
    if (readMem(data, size) < 0 || transformImage() < 0)
    {
        return -1;
    }
    return writeImage();
}

int
BLImageTransformer::transformImage()
{
    int clientH = mClientInfoP->getUsableScreenWidth();
    int clientV = mClientInfoP->getUsableScreenHeight();
    if (mImageP->columns > mImageP->rows)

```

```

    if (mImageP->columns > clientH)
    {
        mImageP->newImage = ScaleImage(mImageP, clientH,
            mImageP->rows * clientH / mImageP->columns);
    }
    if (newImage == NULL)
    {
        //logError("Out of memory when scaling");
        return -1;
    }
    DestroyImage(mImageP);
    mImageP = newImage;
}
else
{
    if (mImageP->rows > clientV)
    {
        Image *newImage = ScaleImage(mImageP, mImageP->columns * clientV / mImageP->rows,
            clientV);
    }
    if (newImage == NULL)
    {
        //logError("Out of memory when scaling");
        return -1;
    }
    DestroyImage(mImageP);
    mImageP = newImage;
}

// examine client info to determine screen bit depth
// VERY rough new attempt

//Image *mapImageP;
QuantizeInfo qi;
qi.tree_depth = 0;
qi.dither = 0;
qi.colorspace = GRAYColorspace;
switch(mClientInfoP->getBitDepth())
{
    case 4:
        qi.number_colors = 16;
        qi.measure_error = 0;
        QuantizeImage(&qi, mImageP);
        break;
    case 2:
        qi.number_colors = 4;
        qi.measure_error = 0;
        qi.dither = (GetNumberColors(mImageP, NULL) > 16);
        QuantizeImage(&qi, mImageP);
        break;
    return 0;
}

int
BLImageTransformer::writeImage()
{
    //ImageInfo ii;

```

```

//GetImageInfo(&i);
char *blob = NULL;
size_t length = 1;
int err = 0;

switch(mToType)
{
case BLTransformManager::kGIFTType:
    strcpy(mImageP->magick, "GIF");
    blob = (char *) ImageToBlob(&mImageInfo, mImageP, &length);
    if (blob == NULL)
    {
        //ERROR! Don't write TODO : what to do?
        return -1;
    }
    err = mOutputTransportP->init(length, BLTransformManager::kGIFTType);
    if (err != -1)
    {
        err = mOutputTransportP->put(blob, length, true);
        FreeMemory(blob);
        break;
    }
case BLTransformManager::kPalmBMPType:
    err = writePalmBitmap();
    break;
}

return err;

static int redcmp(const void *elem1, const void *elem2)
{
    return ((PixelPacket*)elem1)->red - ((PixelPacket*)elem2)->red;
}

static int greencmp(const void *elem1, const void *elem2)
{
    return ((PixelPacket*)elem1)->green - ((PixelPacket*)elem2)->green;
}

int
BLImageTransformer::writePalmBitmap()
{
    BitmapType bitmap;
    memset(&bitmap, 0, sizeof(BitmapType));

    int bitDepth = mClientInfoP->getBitDepth();
    int width = mImageP->columns;
    int height = mImageP->rows;
    int padding = 0;
    int rowbytes = mImageP->columns * bitDepth / 8;
    if ((mImageP->columns * bitDepth) % 8 != 0)
        rowbytes++;
    if (rowbytes % 4 != 0)
    {
        padding = 4 - (rowbytes % 4);
        rowbytes += padding; // word align rows
    }

    bitmap.width = PALM_WORD(width);
    bitmap.height = PALM_WORD(height);
    bitmap.rowbytes = PALM_WORD(rowbytes);
    bitmap.pixelsize = (Byte) bitDepth;
    bitmap.version = (Byte) 1;

    int datasize = rowbytes * height;

    if (mOutputTransportP->init(datasize + sizeof(bitmap), BLTransformManager::kPalmBMPType)
        < 0 ||
        mOutputTransportP->put((char*) &bitmap, sizeof(bitmap), false) < 0)
        return -1;

    char *datap = new char[datasize];
    if (datap == NULL)
        return -1;

    GetPixelCache(mImageP, 0, 0, mImageP->columns, mImageP->rows);
    IndexPacket *ip = mImageP->indexes;
    if (ip == NULL)
        return -1;

    PixelPacket *cmap = (PixelPacket *) malloc(4 * sizeof(PixelPacket));
    memcpy(cmap, mImageP->colormap, 4 * sizeof(PixelPacket));

    if (cmap == NULL)
        return -1;

    // green holds original index position in the map
    for (int u = 0; u < 4; u++)
    {
        cmap[u].green = u;
    }

    // sort according to gray (red) value
    qsort(cmap, 4, sizeof(PixelPacket), redcmp);

    // blue holds the new index position, which will be the palm output level
    for (int v = 0; v < 4; v++)
    {
        cmap[v].blue = 3 - v;
    }

    // sort back to original mapping
    qsort(cmap, 4, sizeof(PixelPacket), greencmp);

    unsigned int mask = 2 * bitDepth - 1;
    int pixelsPerByte = 8 / bitDepth;
    int pos = 0;

    for (unsigned int i = 0; i < mImageP->rows; i++)
    {
        Byte curByte = 0;
        unsigned int j;
        for (j = 0; j < mImageP->columns; j++)
        {
            curByte |= (cmap[ip[j]].blue & mask);
            if (((j + 1) % pixelsPerByte) == 0)
            {
                datap[pos++] = curByte;
                curByte = 0;
            }
            else
            {
                curByte = curByte << bitDepth;
            }
        }
        if (j % pixelsPerByte != 0)
        {
            curByte = curByte << (8 - bitDepth * (j % pixelsPerByte));
            datap[pos++] = curByte;
        }
        pos += padding;
    }

    int err = mOutputTransportP->put(datap, datasize, true);
    delete [] datap;
    free(cmap);
    return err;
}

BLImageTransformerFactory::~BLImageTransformerFactory()

```

```

}

int
BLImageTransformerFactory::Initialize(BLTransformerManager *managerP)
{
    managerP->registerTransformation(BLTransformerManager::kJPEGType,
        BLTransformerManager::kPalmBMPType, this);
    managerP->registerTransformation(BLTransformerManager::kGIFType,
        BLTransformerManager::kPalmBMPType, this);
    managerP->registerTransformation(BLTransformerManager::kPNGType,
        BLTransformerManager::kPalmBMPType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kPalmBMPType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kPalmBMPType, this);

    managerP->registerTransformation(BLTransformerManager::kJPEGType,
        BLTransformerManager::kGIFType, this);
    managerP->registerTransformation(BLTransformerManager::kGIFType,
        BLTransformerManager::kGIFType, this);
    managerP->registerTransformation(BLTransformerManager::kPNGType,
        BLTransformerManager::kGIFType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kGIFType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kGIFType, this);

    //managerP->registerTransformation(BLTransformerManager::kJPEGType,
    //    BLTransformerManager::kPNGType, this);
    managerP->registerTransformation(BLTransformerManager::kGIFType,
        BLTransformerManager::kPNGType, this);
    managerP->registerTransformation(BLTransformerManager::kPNGType,
        BLTransformerManager::kPNGType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kPNGType, this);
    managerP->registerTransformation(BLTransformerManager::kWBMPTType,
        BLTransformerManager::kPNGType, this);

    return 0;
}

const char*
BLImageTransformerFactory::getName()
{
    return "BLImageTransformerFactory";
}

BLTransformation*
BLImageTransformerFactory::makeTransformation(BLTransformation::ContentType fromType,
        BLTransformation::ContentType toType,
        BLClientInfo* infoP,
        BLTransportStream *streamP)
{
    BLImageTransformer *transformerP;

    transformerP = new BLImageTransformer(fromType, toType);

    if (transformerP == NULL)
    {
        return NULL;
    }

    if (transformerP->Init(streamP, this) < 0)
    {
        delete transformerP;
        return NULL;
    }

    transformerP->setClientInfo(infoP);
    return transformerP;
}

// server.cpp

#include "ace/Service_Config.h"
#include "BlHttpAcceptor.h"
#include <stdio.h>

#ifdef ACE_WIN32

```

```

class NullHandler : public ACE_Event_Handler
{
public:
    virtual int handle_signal (int signum, siginfo_t * = 0, ucontext_t * = 0)
    {
        return 0;
    }
};

#ifdef ACE_WIN32
int
main (int argc, char *argv[])
{
    cout << "Starting BLServer..." << endl;

    // Create an adapter to end the event loop.
    ACE_Sig_Adapter sa ((ACE_Sig_Handler_Ex) ACE_Reactor::end_event_loop);

    ACE_Sig_Set sig_set;
    sig_set.sig_add (SIGINT);
    sig_set.sig_add (SIGQUIT);

    // Register ourselves to receive signals so we can shut down
    // gracefully.
    if (ACE_Reactor::instance ()->register_handler (sig_set,
        &sa) == -1)
    {
        ACE_ERROR_RETURN ((LM_ERROR,
            "%p\n"),
            -1);

        // null handler for sigpipe
        #ifndef ACE_WIN32
        NullHandler nullH;
        if (ACE_Reactor::instance ()->register_handler (SIGPIPE,
            &nullH) == -1)
        {
            ACE_ERROR_RETURN ((LM_ERROR,
                "%p\n"),
                -1);
        }
        #endif
    }

    // Try to link in the svc.conf entries dynamically.
    //if (ACE_Service_Config::open (argc, argv) == -1)
    {
        if (errno != ENOENT)
            ACE_ERROR_RETURN ((LM_ERROR,
                "%p\n",
                "open"),
                1);

        else // Use static linking.
        {
            if (ACE::debug () == 0)
                ACE_Log_Msg::disable_debug_messages ();

            // Calling ACE_SVC_INVOKE to create a new Service_Object.
            // Stash the newly created Service_Object into an
            // ACE_Service_Object_Ptr which is an <auto_ptr> specialized
            // for ACE_Service_Object.

            /*char *l_argv[3];
            char configfile[] = "-f "; // MAL_DEFAULT_CONFIG_FILE_NAME;

            l_argv[0] = configfile;
            l_argv[1] = 0;
            ACE_Service_Object_Ptr sp_1 = ACE_SVC_INVOKE (BlHttpAcceptor);

            if (sp_1->init (0, NULL) == -1)
                ACE_ERROR ((LM_ERROR,
                    "%p\n",
                    "MALServer",
                    1));

            // Run forever, performing the configured services until we
            // are shut down by a SIGINT/SIGQUIT signal.
            cout << "BLServer started." << endl;
        }
    }
}

```

```

    ACE_Reactor::run_event_loop ();
    // Destructors of ACE_Service_Object_Ptr's automatically
    // call fini().
}
else // Use dynamic linking.
{
    if (ACE::debug () == 0)
        ACE_Log_Msg::disable_debug_messages ();

    // Run forever, performing the configured services until we are
    // shut down by a SIGINT/SIGQUIT signal.
    cout << "BLServer started." << endl;
    ACE_Reactor::run_event_loop ();
}

return 0;
}
// BLDocNormalizer.cpp
#include "BLDocNormalizer.h"

#ifdef ACE_WIN32
#pragma warning(disable:4786)
#endif

BLDocNormalizer::BLDocNormalizer(BLDocStream *streamP)
: mDocStreamP(streamP),
  mDocTypeP(NULL)
{
}

int
BLDocNormalizer::beginDoc (BLDocType *docTypeP)
{
    mDocTypeP = docTypeP;

    mContextVector.push_back(new Context);
    return mDocStreamP->beginDoc(docTypeP);
}

int
BLDocNormalizer::endDoc()
{
    // close all elements
    while (mContextVector.size() > 0)
    {
        Context *contextP = mContextVector.back();
        while (contextP->mOpenElems.size() > 0)
        {
            if (mDocStreamP->endElement(contextP->mOpenElems.back().getToken() < 0)
                return -1;
            contextP->mOpenElems.pop_back();
        }
        delete contextP;
        mContextVector.pop_back();
    }
    return mDocStreamP->endDoc();
}

int
BLDocNormalizer::charData(const char* data, int length)
{
    mCDATA = data;
    mCDatLength = length;
    mElement.SetTok(BLDocType::KCDATAElement);

    return beginElement_i(smElement);
}

int
BLDocNormalizer::beginElement(BLElement *elemP)
{
    // validate attrs

```

```

    BLElementDef *defP = mDocTypeP->getElementDef(elemP->getToken());
    if (!defP->validateElement(elemP))
        return beginElement_i(elemP);

    int
    BLDocNormalizer::beginElement_i(BLElement *elemP)
    {
        if (mContextVector.size() == 0)
            return -1;
        Context *curContextP = mContextVector.back();
        BLElementDef *defP = mDocTypeP->getElementDef(elemP->getToken());
        if (!defP->getEmptyOK())
        {
            // check to see if it is already pending
            if (mPendingElems.size() > 0)
            {
                for (ElemVector::iterator i = mPendingElems.begin();
                    i != mPendingElems.end();
                    i++)
                {
                    if (i->getToken() == elemP->getToken())
                    {
                        mPendingElems.erase(i, mPendingElems.end());
                        break;
                    }
                }
            }
            mPendingElems.push_back(*elemP);
            return 0;
        }
        else if (mPendingElems.size() > 0)
        {
            // open them all up
            for (ElemVector::iterator i = mPendingElems.begin();
                i != mPendingElems.end();
                i++)
            {
                BLElement *pendingElemP = i;
                openElement(pendingElemP);
            }
        }
        // if we have elements waiting to be opened
        if (curContextP->mClosedElems.size() > 0)
        {
            int backTok = curContextP->mClosedElems.back().getToken();
            if (curContextP->mOpenElems.size() > 0)
            {
                if (defP->getShortestPath(backTok) != NULL)
                {
                    if (openElement(elemP) < 0)
                        return -1;
                    return openElems();
                }
            }
            defP = mDocTypeP->getElementDef(curContextP->mOpenElems.back().getToken());
            if (defP->getShortestPath(backTok) != NULL)
            {
                if (openElems() < 0)
                    return -1;
                return openElement(elemP);
            }
        }
        return openElement(elemP);
    }

    int
    BLDocNormalizer::openElement(BLElement *elemP)
    {

```



```

Context *curContextP = mContextVector.back();
BLElementDef *defP = mDocTypeP->getElementDef(elemP->getTok());
if (defP->isEmpty())
    elemP->setEmpty(true);

// handle root case
if (curContextP->mOpenElems.size() == 0)
{
    if (curContextP == mContextVector.front())
    {
        BLElementDef *rootP = mDocTypeP->getRootElemDef();
        if (elemP->getTok() != rootP->getTok())
        {
            mElement.setTok(rootP->getTok());
            curContextP->mOpenElems.push_back(mElement);
            if (mDocStreamP->beginElement(&mElement) < 0)
                return -1;
        }
        else
            return -1;
    }

    // check if element already open in this context, if so close it
    for (ElemVector::reverse_iterator ri = curContextP->mOpenElems.rbegin();
         ri != curContextP->mOpenElems.rend();
         ri++)
    {
        if (ri->getTok() == elemP->getTok())
            return reopen(elemP);
    }

    // check to see if we can get there from here
    BLElementDef *topElemP =
        mDocTypeP->getElementDef(curContextP->mOpenElems.back().getTok());
    topElemP = topElemP->getShortestPath(elemP->getTok());
    if (topElemP == NULL)
        return 0;

    // traverse the path
    while (topElemP->getTok() != elemP->getTok())
    {
        mElement.setTok(topElemP->getTok());
        mDocStreamP->beginElement(&mElement);
        if (topElemP->hasNewContext())
        {
            curContextP = new Context;
            mContextVector.push_back(curContextP);
        }
        curContextP->mOpenElems.push_back(mElement);
        topElemP = topElemP->getShortestPath(elemP->getTok());
    }

    if (defP->hasNewContext())
    {
        curContextP = new Context;
        mContextVector.push_back(curContextP);
    }

    // check to see if it should be empty
    if (elemP->isEmpty())
    {
        curContextP->mOpenElems.push_back(*elemP);
    }

    if (elemP->getTok() == BLEDocType::KCDATAElement)
        return mDocStreamP->charData(mCDATA, mCDATALength);
    else
        return mDocStreamP->beginElement(elemP);
}

```

```

int
BLEDocNormalizer::reopen(BLElement *elemP)
{
    if (mContextVector.size() == 0)
        return 0;
    if (mPendingElems.size() > 0)
    {
        for (ElemVector::iterator i = mPendingElems.begin();
             i != mPendingElems.end();
             i++)
        {
            if (i->getTok() == elemP->getTok())
            {
                mPendingElems.erase(i, mPendingElems.end());
                return 0;
            }
        }
    }

    Context *curContextP = mContextVector.back();

    // check that it is open
    bool open = false;
    ElemVector::reverse_iterator ri;
    for (ri = curContextP->mOpenElems.rbegin();
         ri != curContextP->mOpenElems.rend();
         ri++)
    {
        if ((*ri)->getTok() == elemP->getTok())
        {
            open = true;
            break;
        }
    }

    if (!open)
        return 0;

    if (closeElem(elemP->getTok() < 0))
        return -1;

    return 0;
}

int
BLEDocNormalizer::reopen(BLElement *elemP)
{
    int elemTok = elemP->getTok();

    // close it and elements above it
    ElemVector::reverse_iterator ri;
    for (ri = curContextP->mOpenElems.rbegin();
         ri != curContextP->mOpenElems.rend();
         ri++)
    {
        if (mDocStreamP->endElement((*ri)->getTok() < 0))
            return -1;

        if ((*ri)->getTok() == elemTok)
            break;
    }

    // set it to new element
    *ri = *elemP;

    // reopen everything
    for (ElemVector::iterator i = ri;
         i != curContextP->mOpenElems.end();
         i++)
    {
        BLElement *openElemP = i;
        if (mDocStreamP->beginElement(openElemP) < 0)
            return -1;
    }
}

```

```

    return 0;
}

int
BLDocNormalizer::closeElem(int elemTok)
{
    ElemVector *curContextP = mContextVector.back();
    bool hardClose = mDocTypeP->getElementDef(elemTok)->hasHardClose();

    // close it and elements above it
    for (ElemVector::reverse_iterator ri = curContextP->mOpenElems.rbegin();
         ri != curContextP->mOpenElems.rend();
         ri++)
    {
        if (mDocStream->endElement(ri->getToken()) < 0)
            return -1;
        curContextP->mOpenElems.pop_back();

        if (ri->getToken() == elemTok)
        {
            break;
        }
        else if (!hardClose)
        {
            BLElement *elemP = ri;
            curContextP->mClosedElems.push_back(*elemP);
        }
    }

    // if end of context, close it out
    if (mDocTypeP->getElementDef(elemTok)->hasNewContext())
    {
        delete curContextP;
        mContextVector.pop_back();
    }

    return 0;
}

int
BLDocNormalizer::openElems()
{
    ElemVector *curContextP = mContextVector.back();

    // if element immediately above it is valid in new element, reopen it, etc
    for (ElemVector::reverse_iterator ri = curContextP->mClosedElems.rbegin();
         ri != curContextP->mClosedElems.rend();
         ri++)
    {
        BLElement *elemP = ri;
        if (openElement(elemP) < 0)
            return -1;
    }

    curContextP->mClosedElems.clear();

    return 0;
}

// BLDocTransformer.cpp
#include "BLDocTransformer.h"
#include "BLXmlParser.h"
#include "BLDocNormalizer.h"
#include "BLXmlOutput.h"
#include "BLWbxmlOutput.h"
#include "BLHtml2WmlAdapter.h"

#ifdef ACE_WIN32
#pragma warning(disable:4786)
#endif

BLHtmlDocType* BLDocTransformer::sHtmlDocType = NULL;
BLWmlDocType* BLDocTransformer::sWmlDocType = NULL;
BLWbxmlTable* BLDocTransformer::sWbxmlTable = NULL;

```

```

BLDocTransformer::BLDocTransformer(BLTransformerManager::ContentType fromType,
                                   BLTransformerManager::ContentType toType)
{
    BLTransformation *transformation = new BLTransformation(
        mNormalizerP(NULL),
        mAdapterP(NULL),
        mOutputP(NULL)
    );

    BLDocTransformer::BLDocTransformer()
    {
        delete mParserP;
        delete mNormalizerP;
        delete mAdapterP;
        delete mOutputP;

        int
        BLDocTransformer::init(BLTransportStream *streamP)
        {
            if (mToType == BLTransformerManager::KWMLType)
            {
                ACE_NEW_RETURN(mOutputP,
                               BLXmlOutput(streamP, mToType),
                               -1);
            }
            else if (mToType == BLTransformerManager::KWMLType)
            {
                ACE_NEW_RETURN(mOutputP,
                               BLWbxmlOutput(streamP, sWbxmlTable, mToType),
                               -1);
            }
            else
            {
                return -1;
            }

            ACE_NEW_RETURN(mNormalizerP,
                           BLDocNormalizer(mOutputP),
                           -1);

            if (mFromType == BLTransformerManager::KWMLType)
            {
                ACE_NEW_RETURN(mParserP,
                               BLXmlParser(sWmlDocType, mNormalizerP),
                               -1);
            }
            else if (mFromType == BLTransformerManager::KHTMLType)
            {
                ACE_NEW_RETURN(mParserP,
                               BLHtmlParser(sHtmlDocType, mNormalizerP),
                               -1);
            }
            else
            {
                return -1;
            }

            return 0;
        }

        BLTransportStream*
        BLDocTransformer::getTransportStream()
        {
            return mParserP;
        }

        int
        BLDocTransformerFactory::initialize(BLTransformerManager *managerP)
        {
            managerP->registerTransformation(BLTransformerManager::KWMLType,
                                             BLTransformerManager::KWMLType, this);
            managerP->registerTransformation(BLTransformerManager::KHTMLType,
                                             BLTransformerManager::KHTMLType, this);
            managerP->registerTransformation(BLTransformerManager::KWMLType,
                                             BLTransformerManager::KWMLType, this);
        }
    }
}

```

```

managerP->registerTransformation(BLTransformManager::KHTMLType,
BLTransformManager::KHTMLType, this);
managerP->registerTransformation(BLTransformManager::KHTMLType,
BLTransformManager::KHTMLType, this);

if (mHtmlDocType.initialize() < 0 ||
    mWmlDocType.initialize() < 0 ||
    mWmlcTable.initialize() < 0)
    return -1;

BLDocTransformer::sHtmlDocType = &mHtmlDocType;
BLDocTransformer::sWmlDocType = &mWmlDocType;
BLDocTransformer::sWmlcTable = &mWmlcTable;

return 0;

}

const char*
BLDocTransformerFactory::getName()
{
    return "BLDocTransformer";
}

BLTransformation*
BLDocTransformerFactory::makeTransformation(BLTransformation::ContentType fromType,
BLTransformation::ContentType toType,
BLClientInfo* /*info*/,
BLTransportStream* stream)
{
    BLDocTransformer* transformP;
    ACE_NEW_RETURN(transformP,
        BLDocTransformer(fromType, toType),
        NULL);
    if (transformP->init(stream) < 0)
    {
        delete transformP;
        return NULL;
    }
    return transformP;
}

// BLDocType.cpp
#include "BLDocType.h"
#define kMaxAttrs 10

BLElementDef::BLElementDef()
{
    minPath(false),
    mContext(false),
    mHardClose(false),
    mIgnoreWhitespace(false),
    mPreserveWhitespace(false),
    mResetWhitespace(false),
    mClass(0),
    mNumAttrs(0),
    mEmptyOK(true),
    mElemPathP(NULL),
    mElemDistancesP(NULL),
    mAttrP(NULL),
    mTok(0)
}

BLElementDef::~BLElementDef()
{
    delete[] mElemPathsP;
    delete[] mAttrP;
    delete[] mAttrIndexesP;
}

int
BLElementDef::init(int elemTok, int numElems, int numAttrs)
{
    mTok = elemTok;

```

```

        mElemPathsP[elemP->getTok()] = *i;
    }
}

minPath = false;
if (minPath < 0)
{
    mElemDistancesP[elemP->getTok()] = -2;
    return -2;
}

mElemDistancesP[elemP->getTok()] = minPath + 1;
return minPath + 1;
}

BLDocType::BLDocType(int numElems)
: mNumElements(numElems),
  mElementDefs(NULL),
  mElementStrings(NULL),
  mAttributeStrings(NULL),
  mValueStrings(NULL)
{
    BLDocType::~BLDocType()
    {
        delete[] mElementDefs;
        delete[] mElementStrings;
        delete[] mAttributeStrings;
        delete[] mValueStrings;
    }

    for (std::vector<BLAttributeDef*>::iterator i = mAttributeDefs.begin();
         i != mAttributeDefs.end();
         i++)
    {
        BLAttributeDef *attrP = *i;
        delete attrP;
    }
}

int
BLDocType::initTables(int numElems, int numAttrs, int numValues)
{
    ACE_NEW_RETURN(mElementDefs,
                   BLElementDef[numElems],
                   -1);
    ACE_NEW_RETURN(mElementStrings,
                   char* [numElems],
                   -1);
    ACE_NEW_RETURN(mAttributeStrings,
                   char* [numAttrs],
                   -1);
    ACE_NEW_RETURN(mValueStrings,
                   char* [numValues],
                   -1);

    ACE_OS::memset(mElementStrings, 0, numElems * sizeof(char*));
    ACE_OS::memset(mAttributeStrings, 0, numAttrs * sizeof(char*));
    ACE_OS::memset(mValueStrings, 0, numValues * sizeof(char*));

    for (int i=0; i<numElems; i++)
    {
        if (mElementDefs[i].init(1, numElems, numAttrs) < 0)
            return -1;
    }

    return 0;
}

int
BLDocType::initPaths()
{
    int i;
    BLElementDef *rootElemP = getRootElemDef();
    for (i=0; i<mNumElements; i++)
    {
        ACE_NEW_RETURN(mElementDefs[i].mElemDistancesP,
                       int [mNumElements],
                       -1);
        ACE_OS::memset(mElementDefs[i].mElemDistancesP, -1, mNumElements * sizeof(int));
    }

    for (i=0; i<mNumElements; i++)
    {
        rootElemP->initShortestPath(&mElementDefs[i]);
    }

    for (i=0; i<mNumElements; i++)
    {
        delete[] mElementDefs[i].mElemDistancesP;
    }
    return 0;
}

void
BLDocType::addClassToElem(int elem, int elemClass)
{
    for (int i=0; i<mNumElements; i++)
    {
        if (mElementDefs[i].mClass & elemClass)
            mElementDefs[i].addElement(&mElementDefs[i]);
    }
}

void
BLDocType::addClassToClass(int superClass, int subclass)
{
    for (int i=0; i<mNumElements; i++)
    {
        if (mElementDefs[i].mClass & subclass)
            mElementDefs[i].addClass(superClass);
    }
}

// adds all the elems in elemClass to all the elems in toClass
void
BLDocType::addElemsToClass(int toClass, int elemClass)
{
    for (int i=0; i<mNumElements; i++)
    {
        if (mElementDefs[i].mClass & toClass)
        {
            for (int j=0; j<mNumElements; j++)
            {
                if (mElementDefs[j].mClass & elemClass)
                    mElementDefs[i].addElement(&mElementDefs[j]);
            }
        }
    }
}

void
BLDocType::addElementTok(char* string, int tok)
{
    mElementStrings[tok] = string;
    mElementToks.bindCaselessToken(string, tok);
}

void
BLDocType::addAttributeTok(char* string, int tok)
{
    mAttributeStrings[tok] = string;
    mAttributeToks.bindCaselessToken(string, tok);
}

void
BLDocType::addValueTok(char* string, int tok)
{
    mValueStrings[tok] = string;
}

```

```

    mValueToks.bindCaselessToken(string, tok);
}

void
BLDocType::addEntityTok(char* string, int tok)
{
    mEntityToks.bindToken(string, tok);
}

BLAttributeDef*
BLDocType::makeAttrDef(int attrTok, BLAttribute::Type attrType, bool required)
{
    BLAttributeDef *defp = new BLAttributeDef(attrTok, attrType, required, mValueToks);
    mAttributeDefs.push_back(defp);
    return defp;
}

bool
BLElementDef::validateElement(BLElement *elem)
{
    bool haveAttrs[kMaxAttrs];
    ACE_OS::memset(haveAttrs, 0, kMaxAttrs * sizeof(bool));
    BLElement::AttrVector *attrSP = elem->getAttrVector();
    BLElement::AttrVector::iterator i = attrSP->begin();
    while (i != attrSP->end())
    {
        BLAttributeDef *attrDefP = mAttrSP[i->getTok()];
        if (attrDefP == NULL ||
            haveAttrs[mAttrIndexesP[i->getTok()]] == true ||
            !attrDefP->verifyValue(*i))
        {
            attrSP->erase(i);
        }
        else
        {
            haveAttrs[mAttrIndexesP[i->getTok()]] = true;
            i++;
        }
    }
}

bool haveRequireds = true;
if (mRequiredAttrs.size() > 0)
{
    for (std::vector<BLAttributeDef*>::iterator i = mRequiredAttrs.begin();
         i != mRequiredAttrs.end();
         i++)
    {
        if (!haveAttrs[mAttrIndexesP[*i]->getTok()])
        {
            return false;
        }
    }
    return true;
}

bool
BLAttributeDef::verifyValue(BLAttribute *attrP)
{
    if (mType == attrP->getType())
    {
        return true;
    }
    if (attrP->getType() != BLAttribute::KCDATAtype)
    {
        return false;
    }
    attrP->setType(mType);
    char *cdata = attrP->getCDATAvalue();
    switch (mType)
    {
        case BLAttribute::kTokenType:
        {
            int tok = mValueTokTableP->getCaselessToken(cdata);

```

```

    if (tok == -1)
    {
        return false;
    }
    if (i != mValueToks.end())
    {
        if (*i == tok)
        {
            attrP->setTokValue(tok);
            return true;
        }
    }
    break;
}

case BLAttribute::kIntType:
{
    if (cdata[0] == '\0')
    {
        return false;
    }
    char *endPtr;
    int value = ACE_OS::strtol(cdata, &endPtr, 10);
    if (*endPtr == '\0')
    {
        attrP->setIntValue(value);
        return true;
    }
    break;
}

case BLAttribute::kBoolType:
{
    if (ACE_OS::strcasecmp(cdata, "true") == 0)
    {
        attrP->setBoolValue(true);
        return true;
    }
    else if (ACE_OS::strcasecmp(cdata, "false") == 0)
    {
        attrP->setBoolValue(false);
        return true;
    }
    break;
}

case BLAttribute::kLengthType:
{
    int value = ACE_OS::atoi(cdata);
    if (value == 0)
    {
        return false;
    }
    if (ACE_OS::strchr(cdata, '%') != NULL)
    {
        attrP->setIntValue(-value);
    }
    else
    {
        attrP->setIntValue(value);
        return true;
    }
    break;
}

case BLAttribute::kNoValueType:
{
    return true;
}

return false;
}

// BLHtmlDocType.cpp
#include "BLHtmlDocType.h"

enum {
    kFontClass = 1,
    kPhraseClass = 2^1,
    kSpecialClass = 2^2,
    kFormClass = 2^3,
    kTextClass = 2^4,
    kBlockClass = 2^5,
    kFlowClass = 2^6,
    kListClass = 2^7,
    kBodyContentClass = 2^8,

```

```

kHeadMiscClass = 2^9,
kHeadClass = 2^10,
};

BLHtmlDocType::BLHtmlDocType()
{
    : BLDocType(kNumElements)
}

int
BLHtmlDocType::Initialize()
{
    if (initTables(kNumElements, kNumAttrs, kNumValues) < 0)
        return -1;

    initClasses();
    initElements();
    if (initPaths() < 0)
        return -1;

    initElementToks();
    initAttributeToks();
    initValueToks();
    initEntityToks();

    return 0;
}

void
BLHtmlDocType::initClasses()
{
    // head_misc
    mElementDefs[kScriptElement] .addToClass(kHeadMiscClass);
    mElementDefs[kStyleElement] .addToClass(kHeadMiscClass);
    mElementDefs[kMetaElement] .addToClass(kHeadMiscClass);

    // heading
    mElementDefs[kHeading1Element] .addToClass(kHeadingClass);
    mElementDefs[kHeading2Element] .addToClass(kHeadingClass);
    mElementDefs[kHeading3Element] .addToClass(kHeadingClass);
    mElementDefs[kHeading4Element] .addToClass(kHeadingClass);
    mElementDefs[kHeading5Element] .addToClass(kHeadingClass);
    mElementDefs[kHeading6Element] .addToClass(kHeadingClass);

    // list
    mElementDefs[kUnorderedListElement] .addToClass(kListClass);
    mElementDefs[kOrderedListElement] .addToClass(kListClass);
    mElementDefs[kOrderedListElement] .addToClass(kListClass);
    mElementDefs[kListElement] .addToClass(kListClass);

    // font
    mElementDefs[kFontElement] .addToClass(kFontClass);
    mElementDefs[kItalicElement] .addToClass(kFontClass);
    mElementDefs[kBoldElement] .addToClass(kFontClass);
    mElementDefs[kUnderlineElement] .addToClass(kFontClass);
    mElementDefs[kStrikeElement] .addToClass(kFontClass);
    mElementDefs[kBigElement] .addToClass(kFontClass);
    mElementDefs[kSmallElement] .addToClass(kFontClass);
    mElementDefs[kSubscriptElement] .addToClass(kFontClass);
    mElementDefs[kSuperscriptElement] .addToClass(kFontClass);

    // phrase
    mElementDefs[kEmphasisElement] .addToClass(kPhraseClass);
    mElementDefs[kStrongElement] .addToClass(kPhraseClass);

    // special
    mElementDefs[kAnchorElement] .addToClass(kSpecialClass);
    mElementDefs[kImageElement] .addToClass(kSpecialClass);
    mElementDefs[kFontElement] .addToClass(kSpecialClass);
    mElementDefs[kLinebreakElement] .addToClass(kSpecialClass);
    mElementDefs[kScriptElement] .addToClass(kSpecialClass);
    mElementDefs[kMapElement] .addToClass(kSpecialClass);

    // form
    mElementDefs[kInputElement] .addToClass(kFormClass);

    mElementDefs[kSelectElement] .addToClass(kFormClass);
    mElementDefs[kTextAreaElement] .addToClass(kFormClass);

    mElementDefs[kDataElement] .addToClass(kTextClass);
    addClassToClass(kTextClass, kFontClass);
    addClassToClass(kTextClass, kPhraseClass);
    addClassToClass(kTextClass, kFormClass);

    // block
    mElementDefs[kParagraphElement] .addToClass(kBlockClass);
    mElementDefs[kPreElement] .addToClass(kBlockClass);
    mElementDefs[kDefinitionListElement] .addToClass(kBlockClass);
    mElementDefs[kDivElement] .addToClass(kBlockClass);
    mElementDefs[kCenterElement] .addToClass(kBlockClass);
    mElementDefs[kBlockQuoteElement] .addToClass(kBlockClass);
    mElementDefs[kFormElement] .addToClass(kBlockClass);
    mElementDefs[kHorizontalRuleElement] .addToClass(kBlockClass);
    mElementDefs[kTableElement] .addToClass(kBlockClass);
    addClassToClass(kBlockClass, kListClass);

    // flow
    addClassToClass(kFlowClass, kTextClass);
    addClassToClass(kFlowClass, kBlockClass);

    // body_content
    addClassToClass(kBodyContentClass, kHeadingClass);
    addClassToClass(kBodyContentClass, kTextClass);
    addClassToClass(kBodyContentClass, kBlockClass);

    // font_class
    addElementToClass(kFontClass, kTextClass);

    // phrase_class
    addElementToClass(kPhraseClass, kTextClass);

    void
    BLHtmlDocType::initElements()
    {
        BLAttributeDef *cdataSizeImp = makeAttrDef(kSizeAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdataSizeReq = makeAttrDef(kSizeAttr, BLAttribute::kDataType,
            true);
        BLAttributeDef *cdataColorImp = makeAttrDef(kColorAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdataBgColorImp = makeAttrDef(kBackgroundAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdataTextImp = makeAttrDef(kTextAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdatalinkImp = makeAttrDef(kLinkAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdatalinkImp = makeAttrDef(kVisitedLinkAttr,
            BLAttribute::kDataType, false);
        BLAttributeDef *cdatalinkImp = makeAttrDef(kActiveLinkAttr,
            BLAttribute::kDataType, false);
        BLAttributeDef *cdatalinkImp = makeAttrDef(kAlignAttr, BLAttribute::kType,
            false);
        tokTAlignImp->addValueTok(kLeftValue);
        tokTAlignImp->addValueTok(kCenterValue);
        tokTAlignImp->addValueTok(kRightValue);
        BLAttributeDef *cdatanameImp = makeAttrDef(kNameAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdatanameImp = makeAttrDef(kHrefAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdatanameImp = makeAttrDef(kHrefAttr, BLAttribute::kDataType,
            true);
        BLAttributeDef *cdatashapeImp = makeAttrDef(kShapeAttr, BLAttribute::kType, false);
        tokShapeImp->addValueTok(kRectValue);
        tokShapeImp->addValueTok(kCircleValue);
        tokShapeImp->addValueTok(kPolyValue);
        BLAttributeDef *cdatcoordsImp = makeAttrDef(kCoordsAttr, BLAttribute::kDataType,
            false);
        BLAttributeDef *cdatcoordsImp = makeAttrDef(kHrefAttr, BLAttribute::kType,
            false);
    }
}

```

```

BLAttributeDef *cdDataAltImp = makeAttrDef(kAltAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *cdDataSrcReq = makeAttrDef(kSrcAttr, BLAttribute::KCDDataType, true);
BLAttributeDef *cdDataSrcImp = makeAttrDef(kSrcAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *tokAlignImp = makeAttrDef(kAlignAttr, BLAttribute::KTokenType, false);
tokAlignImp->addValueTok(kTopValue);
tokAlignImp->addValueTok(kMiddleValue);
tokAlignImp->addValueTok(kBottomValue);
tokAlignImp->addValueTok(kLeftValue);
tokAlignImp->addValueTok(kRightValue);
BLAttributeDef *intHeightImp = makeAttrDef(kHeightAttr, BLAttribute::KIntType, false);
BLAttributeDef *intWidthImp = makeAttrDef(kWidthAttr, BLAttribute::KIntType, false);
BLAttributeDef *intBorderImp = makeAttrDef(kBorderAttr, BLAttribute::KIntType, false);
BLAttributeDef *intSpaceImp = makeAttrDef(kHorizontalSpaceAttr, BLAttribute::KIntType, false);
BLAttributeDef *intSpaceImp = makeAttrDef(kVerticalSpaceAttr, BLAttribute::KIntType, false);
BLAttributeDef *intTypeImp = makeAttrDef(kTypeAttr, BLAttribute::KLengthType, false);
BLAttributeDef *intStartImp = makeAttrDef(kStartAttr, BLAttribute::KIntType, false);
BLAttributeDef *cdDataActionImp = makeAttrDef(kActionAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *cdDataMethodImp = makeAttrDef(kMethodAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *cdDataEndTypeImp = makeAttrDef(kEncodingTypeAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *tokTypeImp = makeAttrDef(kTypeAttr, BLAttribute::KTokenType, false);
tokTypeImp->addValueTok(kTextValue);
tokTypeImp->addValueTok(kPasswordValue);
tokTypeImp->addValueTok(kCheckboxValue);
tokTypeImp->addValueTok(kRadioValue);
tokTypeImp->addValueTok(kSubmitValue);
tokTypeImp->addValueTok(kResetValue);
tokTypeImp->addValueTok(kFileValue);
tokTypeImp->addValueTok(kHiddenValue);
tokTypeImp->addValueTok(kImageValue);
BLAttributeDef *cdDataValueImp = makeAttrDef(kValueAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *noValueCheckedImp = makeAttrDef(kCheckedAttr, BLAttribute::KNoValueType, false);
BLAttributeDef *intMaxLengthImp = makeAttrDef(kMaxLengthAttr, BLAttribute::KIntType, false);
BLAttributeDef *noValueMultipleImp = makeAttrDef(kMultipleAttr, BLAttribute::KNoValueType, false);
BLAttributeDef *noValueSelectedImp = makeAttrDef(kSelectedAttr, BLAttribute::KNoValueType, false);
BLAttributeDef *intRowsImp = makeAttrDef(kRowsAttr, BLAttribute::KIntType, false);
BLAttributeDef *intColsImp = makeAttrDef(kColsAttr, BLAttribute::KIntType, false);
BLAttributeDef *intCellSpacingImp = makeAttrDef(kCellSpacingAttr, BLAttribute::KIntType, false);
BLAttributeDef *intCellPaddingImp = makeAttrDef(kCellPaddingAttr, BLAttribute::KIntType, false);
BLAttributeDef *tokAlignImp = makeAttrDef(kAlignAttr, BLAttribute::KTokenType, false);
tokAlignImp->addValueTok(kTopValue);
tokAlignImp->addValueTok(kMiddleValue);
tokAlignImp->addValueTok(kBottomValue);
tokAlignImp->addValueTok(kLeftValue);
tokAlignImp->addValueTok(kRightValue);
tokAlignImp->addValueTok(kBottomValue);
BLAttributeDef *intRowSpanImp = makeAttrDef(kRowSpanAttr, BLAttribute::KIntType, false);

```

```

false);
BLAttributeDef *intColSpanImp = makeAttrDef(kColSpanAttr, BLAttribute::KIntType, false);
BLAttributeDef *tokHttpEqvImp = makeAttrDef(kHttpEqvAttr, BLAttribute::KTokenType, false);
tokHttpEqvImp->addValueTok(kRefreshValue);
BLAttributeDef *cdDataContentReq = makeAttrDef(kContentAttr, BLAttribute::KCDDataType, false);
BLAttributeDef *intValueImp = makeAttrDef(kValueAttr, BLAttribute::KIntType, false);
// html
mElementDefs[kHtmlElement].addElement(mElementDefs[kHeadElement]);
mElementDefs[kHtmlElement].addElement(mElementDefs[kBodyElement]);
mElementDefs[kHtmlElement].setIgnoreWhitespace();
// basefont
mElementDefs[kHtmlElement].addAttribute(cdataSizeReq);
// body
addClassToElem(kBodyElement, kBodyContentClass);
mElementDefs[kBodyElement].addAttribute(cdataBgcolorImp);
mElementDefs[kBodyElement].addAttribute(cdataTextImp);
mElementDefs[kBodyElement].addAttribute(cdataLinkImp);
mElementDefs[kBodyElement].addAttribute(cdataLinkImp);
mElementDefs[kBodyElement].addAttribute(cdataLinkImp);
mElementDefs[kBodyElement].setResetWhiteSpace();
// div
addClassToElem(kDivElement, kBodyContentClass);
mElementDefs[kDivElement].addAttribute(cdataAlignImp);
mElementDefs[kDivElement].setEmptyOK(false);
mElementDefs[kDivElement].setResetWhiteSpace();
// a
addClassToElem(kAnchorElement, kTextClass);
mElementDefs[kAnchorElement].addAttribute(cdataNameImp);
mElementDefs[kAnchorElement].addAttribute(cdataHrefImp);
// map
mElementDefs[kMapElement].addElement(mElementDefs[kAreaElement]);
mElementDefs[kMapElement].addAttribute(cdataNameImp);
mElementDefs[kMapElement].setEmptyOK(false);
// area
mElementDefs[kAreaElement].addAttribute(cdataShapeImp);
mElementDefs[kAreaElement].addAttribute(cdataCoordsImp);
mElementDefs[kAreaElement].addAttribute(cdataHrefImp);
mElementDefs[kAreaElement].addAttribute(cdataHrefImp);
mElementDefs[kAreaElement].addAttribute(cdataAltImp);
mElementDefs[kImageElement].addAttribute(cdataSrcReq);
mElementDefs[kImageElement].addAttribute(cdataAltImp);
mElementDefs[kImageElement].addAttribute(cdataAlignImp);
mElementDefs[kImageElement].addAttribute(cdataHeightImp);
mElementDefs[kImageElement].addAttribute(cdataWidthImp);
mElementDefs[kImageElement].addAttribute(cdataBorderImp);
mElementDefs[kImageElement].addAttribute(cdataVspaceImp);
mElementDefs[kImageElement].addAttribute(cdataVspaceImp);
mElementDefs[kImageElement].addAttribute(cdataUseMapImp);
mElementDefs[kImageElement].addAttribute(cdataUseMapImp);
// hr
mElementDefs[kHorizontalRuleElement].addAttribute(cdataAlignImp);
mElementDefs[kHorizontalRuleElement].addAttribute(cdataNoShadeImp);
mElementDefs[kHorizontalRuleElement].addAttribute(cdataAlignImp);
mElementDefs[kHorizontalRuleElement].addAttribute(cdataAlignImp);
// p
addClassToElem(kParagraphElement, kTextClass);
mElementDefs[kParagraphElement].addAttribute(cdataAlignImp);
mElementDefs[kParagraphElement].setEmptyOK(false);
mElementDefs[kParagraphElement].setResetWhiteSpace();
// br
mElementDefs[kLinebreakElement].setResetWhiteSpace();
// headings
addElemToClass(kHeadingClass, kTextClass);
mElementDefs[kHeading1Element].addAttribute(cdataAlignImp);
mElementDefs[kHeading2Element].addAttribute(cdataAlignImp);
mElementDefs[kHeading3Element].addAttribute(cdataAlignImp);
mElementDefs[kHeading4Element].addAttribute(cdataAlignImp);
mElementDefs[kHeading5Element].addAttribute(cdataAlignImp);
mElementDefs[kHeading6Element].addAttribute(cdataAlignImp);

```







```

addEntityTok("cedil", 184);
addEntityTok("supl", 185);
addEntityTok("ordm", 186);
addEntityTok("raquo", 187);
addEntityTok("frac14", 188);
addEntityTok("frac12", 189);
addEntityTok("frac34", 190);
addEntityTok("quest", 191);
addEntityTok("Agrave", 192);
addEntityTok("Aacute", 193);
addEntityTok("Acirc", 194);
addEntityTok("Atilde", 195);
addEntityTok("Auml", 196);
addEntityTok("Aring", 197);
addEntityTok("Aring", 198);
addEntityTok("Ccedil", 199);
addEntityTok("Egrave", 200);
addEntityTok("Eacute", 201);
addEntityTok("Ecirc", 202);
addEntityTok("Euml", 203);
addEntityTok("Igrave", 204);
addEntityTok("Iacute", 205);
addEntityTok("Icirc", 206);
addEntityTok("Iuml", 207);
addEntityTok("ETH", 208);
addEntityTok("Ntilde", 209);
addEntityTok("Ograve", 210);
addEntityTok("Oacute", 211);
addEntityTok("Ocirc", 212);
addEntityTok("Otilde", 213);
addEntityTok("Ouml", 214);
addEntityTok("times", 215);
addEntityTok("Oslash", 216);
addEntityTok("Ugrave", 217);
addEntityTok("Uacute", 218);
addEntityTok("Ucirc", 219);
addEntityTok("Uuml", 220);
addEntityTok("Yacute", 221);
addEntityTok("THORN", 222);
addEntityTok("szlig", 223);
addEntityTok("agrave", 224);
addEntityTok("aacute", 225);
addEntityTok("acirc", 226);
addEntityTok("atilde", 227);
addEntityTok("auml", 228);
addEntityTok("aring", 229);
addEntityTok("cedil", 230);
addEntityTok("egrave", 231);
addEntityTok("eacute", 232);
addEntityTok("ecirc", 233);
addEntityTok("euml", 234);
addEntityTok("eth", 235);
addEntityTok("igrave", 236);
addEntityTok("iacute", 237);
addEntityTok("icirc", 238);
addEntityTok("iuml", 239);
addEntityTok("ieth", 240);
addEntityTok("ntilde", 241);
addEntityTok("ograve", 242);
addEntityTok("ograde", 243);
addEntityTok("ocirc", 244);
addEntityTok("otilde", 245);
addEntityTok("ouml", 246);
addEntityTok("divide", 247);
addEntityTok("oslash", 248);
addEntityTok("ugrave", 249);
addEntityTok("uacute", 250);
addEntityTok("ucirc", 251);
addEntityTok("uuml", 252);
addEntityTok("yacute", 253);
addEntityTok("thorn", 254);
addEntityTok("yuml", 255);

const char*

BLHtmlDocType::getDocTypeString()
{
    BLHtmlDocType *docType = HTML_PUBLIC "-//IETF//DTD HTML//EN";
    return getElementTypeDef(kHtmlElement);
}

BLHtmlDocType::getRootElementDef()
{
    return getElementTypeDef(kHtmlElement);
}

// BLHtmlParser.cpp
/*
#include "BLHtmlParser.h"
#include "BLUtils.h"

BLHtmlParser::BLHtmlParser(BLDocType *docType, BLDocStream *docStream)
: mDocTypeP(docTypeP),
  mDocStreamP(docStreamP)
{
}

void
BLHtmlParser::init(BLDocument* docP)
{
    mParseP = mParseBuf;
    mScratchP = mScratchBuf;
    mValueP = mScratchBuf;
    mState = 1;
    mScratchDistance = 0;
    mLastCharDataSpace = true;

    // returns 0 if we find closeChar before another openChar
    // 1 if we find another openChar before maxDistance or a closeChar before mindistance
    // or nothing by max distance
    // 2 if we reached the end of the buffer before finding either
    inline int
    BLHtmlParser::lookahead(char closeChar,
                           int minDistance,
                           int maxDistance,
                           char *buf,
                           int buflen)
    {
        int distance = buflen < maxDistance ? buflen : maxDistance;
        for (int i=0; i<distance; i++)
        {
            if (buf[i] == closeChar)
            {
                mScratchDistance = 0;
                if (i + (mScratchP - mScratchBuf) < minDistance)
                    return 1;
                return 0;
            }
            if (buf[i] == mLookaheadChar
                || buf[i] == '<') // not reached in '<' case, but needed for '<' case
            {
                mScratchDistance = 0;
                return 1;
            }
            if (buflen < maxDistance)
            {
                ACE_OS::memcpy(mScratchP, buf, distance);
                mScratchP += distance;
                mScratchDistance = maxDistance - distance;
                mScratchMinDistance = minDistance;
                mScratchCloseChar = closeChar;
                return 2;
            }
            return 1;
        }
        return 0;
    }

    inline void

```

```

BLHtmlParser::beginTag()
{
    *mParseP = '\0';
    int elemTok = mDocTypeP->getElementTok(mParseBuf);
    mElement.clear();
    mElement.setTok(elemTok);
    mParseP = mParseBuf;
}

inline void
BLHtmlParser::beginTagDone()
{
    char *value = mScratchBuf;
    if (mElement.getTok() != -1)
        mDocStreamP->beginElement(&mElement);
    *mValueP = mScratchBuf;
}

inline void
BLHtmlParser::endTag()
{
    *mParseP = '\0';
    int elemTok = mDocTypeP->getElementTok(mParseBuf);
    if (elemTok != -1)
        mDocStreamP->endElement(elemTok);
    mParseP = mParseBuf;
}

inline void
BLHtmlParser::attribute()
{
    *mParseP = '\0';
    int attrTok = mDocTypeP->getAttributeTok(mParseBuf);
    if (attrTok != -1)
    {
        mCurAttr.first = attrTok;
        mCurAttr.second.mType = BLDocument::kCDataValue;
        mCurAttr.second.mData.cdataValue = mValueP;
        mAttrs.push_back(mCurAttr);
    }
    mLastValueSpace = true;
    mParseP = mParseBuf;
}

inline void
BLHtmlParser::value()
{
    *mValueP++ = '\0';
}

inline void
BLHtmlParser::charData()
{
    *mParseP = '\0';
    mCurTag = BLDocument::kCDataElement;
    if (mInTitle)
    {
        int len = BLUtils::min(mParseP - mParseBuf, kTitleSize - (mTitleP - mTitleBuf));
        if (len > 0)
        {
            ACE_OS::strncpy(mTitleP, mParseBuf, len);
            mTitleP += len;
        }
        else
        {
            doLinebreaks();
        }
    }
}

if (!mIgnoreCData)
{
    mCurAttr.first = BLDocument::kCDataAttr;
    mCurAttr.second.mType = BLDocument::kCDataValue;
    mCurAttr.second.mData.cdataValue = mParseBuf;
    mAttrs.push_back(mCurAttr);
    mDocP->beginElement(BLDocument::kCDataElement, &mAttrs, true);
}

mLastTag = mCurTag;
mParseP = mParseBuf;

void
BLHtmlParser::dumpScratch()
{
    *mScratchP = '\0';
    if (!mIgnoreCData)
    {
        mAttrs.clear();
        mCurAttr.first = BLDocument::kCDataAttr;
        mCurAttr.second.mType = BLDocument::kCDataValue;
        mCurAttr.second.mData.cdataValue = mScratchBuf;
        mAttrs.push_back(mCurAttr);
        mDocP->beginElement(BLDocument::kCDataElement, &mAttrs, true);
    }
}

inline void
BLHtmlParser::doEntity(int entity)
{
    if (!mIgnoreCData && !mInTitle)
    {
        mCurTag = BLDocument::kCharEntityElement;
        doLinebreaks();
        mAttrs.clear();
        mCurAttr.first = BLDocument::kEntityAttr;
        mCurAttr.second.mType = BLDocument::kIntValue;
        mCurAttr.second.mData.intValue = entity;
        mAttrs.push_back(mCurAttr);
        mDocP->beginElement(BLDocument::kCharEntityElement, &mAttrs, true);
        mLastCharDataSpace = false;
    }
    mLastTag = mCurTag;
}

inline void
BLHtmlParser::charEntity()
{
    *mParseP = '\0';
    int c = sEntityTable.getTokenID(mParseBuf);
    if (c != -1)
        doEntity(c);
    mParseP = mParseBuf;
}

inline void
BLHtmlParser::decimalEntity()
{
    *mParseP = '\0';
    int c = ACE_OS::atoi(mParseBuf);
    if (c != 0)
        doEntity(c);
    mParseP = mParseBuf;
}

inline void
BLHtmlParser::hexEntity()
{
    *mParseP = '\0';
}

```

```
int c = (char) ACE_OS::strtol(mParseBuf, NULL, 16);
```

```
if (c != 0)
    doEntity(c);
```

```
mParseP = mParseBuf;
```

```
inline void
```

```
BLHtmlParser::beginDocument()
```

```
{
    mDocP->documentBegin();
```

```
}
```

```
inline void
```

```
BLHtmlParser::endDocument()
```

```
{
    if ((mState == 1) && (mParseP != mParseBuf))
```

```
        charData();
```

```
    mDocP->documentEnd();
```

```
    mDocP = NULL;
```

```
}
```

```
int
```

```
BLHtmlParser::put(char *data, int size, bool final)
```

```
{
    if (mDocP == NULL || data == NULL)
```

```
        return -1;
```

```
    if (mScratchDistance > 0)
```

```
    {
```

```
        int look = lookahead(mScratchCloseChar, mScratchMinDistance, mScratchDistance, data,
```

```
        size);
```

```
        if (mLookaheadChar == '<')
```

```
        {
```

```
            switch (look)
```

```
            {
```

```
                case 0:
```

```
                    if (mParseP != mParseBuf)
```

```
                        charData();
```

```
                    if (mScratchChar == '/')
```

```
                    {
```

```
                        mState = 10;
```

```
                    }
                    else
```

```
                    {
```

```
                        mState = 3;
```

```
                    }
                    break;
```

```
                case 1:
```

```
                    if (mParseP != mParseBuf)
```

```
                        charData();
```

```
                    doEntity('<');
```

```
                    dumpScratch();
```

```
                    mLastCharDataSpace = false;
```

```
                    break;
```

```
                case 2:
```

```
                    if (final)
```

```
                        endDocument();
```

```
                    return 0;
```

```
                    break;
```

```
            }
        }
        else if (mLookaheadChar == '&')
```

```
        {
```

```
            switch (look)
```

```
            {
```

```
                case 0:
```

```
                    if (mParseP != mParseBuf)
```

```
                        charData();
```

```
                    mState = 13;
```

```
                    break;
```

```
                case 1:
```

```
                    if (mParseP != mParseBuf)
```

```
                        charData();
```

```
                    doEntity('&');
```

```
                }
            }
        }
        return 0;
    }
    if (mScratchDistance > 0)
```

```
    {
```

```
        int look = lookahead(mScratchCloseChar, mScratchMinDistance, mScratchDistance, data,
```

```
        size);
```

```
        if (mLookaheadChar == '<')
```

```
        {
```

```
            switch (look)
```

```
            {
```

```
                case 0:
```

```
                    if (mParseP != mParseBuf)
```

```
                        charData();
```

```
                    if (mScratchChar == '/')
```

```
dumpScratch();
```

```
mLastCharDataSpace = false;
```

```
break;
```

```
case 2:
```

```
if (final)
```

```
{
```

```
    if (mParseP != mParseBuf)
```

```
        charData();
```

```
    doEntity('&');
```

```
    dumpScratch();
```

```
    mLastCharDataSpace = false;
```

```
    }
```

```
    else
```

```
        return 0;
```

```
    break;
```

```
}
```

```
    }
    parseBuf(mScratchBuf, mScratchP - mScratchBuf, false);
```

```
    }
```

```
    parseBuf(data, size, final);
```

```
    return 0;
```

```
}
```

```
// returns 0 normally, 1 if in the middle of lookahead
```

```
void
```

```
BLHtmlParser::parseBuf(char *data, int size, bool final)
```

```
{
```

```
    char *cp;
```

```
    int look;
```

```
    if (size > 0)
```

```
    {
```

```
        for (cp = data;
```

```
            cp != data+size;
```

```
            cp++)
```

```
        {
```

```
            if (*cp > 0x7E) // garbage
```

```
                continue;
```

```
            switch (mState)
```

```
            {
```

```
                case 1: // character data
```

```
                    if (*cp <= 0x20) // whitespace
```

```
                    {
```

```
                        if (!mLastCharDataSpace)
```

```
                        {
```

```
                            *mParseP++ = *cp;
```

```
                            mLastCharDataSpace = true;
```

```
                        }
```

```
                    }
                    break;
```

```
                    switch (*cp)
```

```
                    {
```

```
                        case '<':
```

```
                            mState = 2;
```

```
                            *mScratchP = mScratchBuf;
```

```
                            mLookaheadChar = '<';
```

```
                            look = lookahead('>', 1, kParseBufSize, cp + 1, size - (cp - data) - 1);
```

```
                            switch (look)
```

```
                            {
```

```
                                case 0:
```

```
                                    if (mParseP != mParseBuf)
```

```
                                        charData();
```

```
                                    mState = 2;
```

```
                                    break;
```

```
                                case 1:
```

```
                                    if (mParseP != mParseBuf)
```

```
                                        charData();
```

```
                                    doEntity('<');
```

```
                                    mLastCharDataSpace = false;
```

```
                                    break;
```

```
                                case 2:
```

```
                                    if (final)
```

```
                                        endDocument();
```

```
                                    return;
```

```
                                }
```

```
                            }
```

```
                    }
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```



```

break;
case 5:
// attribute
if (*cp <= 0x20) //whitespace
{
attribute();
mState = 8;
break;
}
switch (*cp)
{
case '>':
beginTagDone();
mState = 1;
break;
case '=':
mState = 6;
break;
case '\n':
attribute();
mState = 7;
break;
case '>':
beginTagDone();
mParsep = mParseBuf;
mState = 1;
break;
default:
*(mParsep++) = CASELESS(*cp);
break;
}
case 6:
// =
if (*cp <= 0x20) //whitespace
break;
switch (*cp)
{
case '\n':
attribute();
mState = 7;
break;
case '>':
beginTagDone();
mState = 1;
break;
default:
attribute();
*(mValuep++) = *cp;
mState = 9;
break;
}
break;
case 7:
// value
if (*cp <= 0x20) //whitespace
{
if (!mLastValueSpace)
{
*(mValuep++) = ' ';
mLastValueSpace = true;
}
break;
}
switch (*cp)
{
case '\n':
value();
mState = 4;
break;
default:
mLastValueSpace = false;
*(mValuep++) = *cp;
break;
}
}
}
// end tag token
if (*cp <= 0x20) //whitespace
{
endTag();
mState = 12;
break;
}
}
}
// end tag begin
if (*cp <= 0x20) //whitespace
{
if (mParsep != mParseBuf)
charData();
doEntity('<');
*(mParsep++) = '/';
*(mParsep++) = ' ';
mLastCharDataSpace = true;
mState = 1;
break;
}
}
switch (*cp)
{
case '>':
mState = 1;
break;
default:
*(mParsep++) = CASELESS(*cp);
mState = 11;
break;
}
break;
case 11:
// end tag token
if (*cp <= 0x20) //whitespace
{
endTag();
mState = 12;
break;
}
}
}

```

```

}
switch (*cp)
{
case '>':
    endtag();
    mState = 1;
    break;
default:
    *(mParsep++) = CASELESS(*cp);
    break;
}
break;

case 12:
    // end tag end
    if (*cp == '>')
        mState = 1;
    break;

case 13:
    // &
    switch (*cp)
    {
    case '#':
        mState = 15;
        break;
    default:
        *(mParsep++) = *cp;
        mState = 14;
        break;
    }
    break;

case 14:
    // character entity
    switch (*cp)
    {
    case ';':
        charEntity();
        mState = 1;
        break;
    default:
        *(mParsep++) = *cp;
        break;
    }
    break;

case 15:
    // decimal/hex entity
    switch (*cp)
    {
    case 'x':
        mState = 17;
        break;
    default:
        *(mParsep++) = *cp;
        mState = 16;
        break;
    }
    break;

case 16:
    // decimal entity
    switch (*cp)
    {
    case ';':
        decimalEntity();
        mState = 1;
        break;
    default:
        *(mParsep++) = *cp;
        break;
    }
    break;

case 17:
    // hex entity
    switch (*cp)
    {
    case ';':
        hexEntity();
        mState = 1;
        break;
    default:
        *(mParsep++) = *cp;
        break;
    }
    break;

case 18:
    // <|
    switch (*cp)
    {
    case '>':
        mState = 1;
        break;
    case '-':
        mState = 19;
        break;
    default:
        mState = 12;
        break;
    }
    break;

case 19:
    // <|-
    switch (*cp)
    {
    case '>':
        mState = 1;
        break;
    case '-':
        mState = 20;
        break;
    default:
        mState = 12;
        break;
    }
    break;

case 20:
    // comment body
    if (*cp == '-')
        mState = 21;
    break;

case 21:
    // -
    if (*cp == '-')
        mState = 22;
    else
        mState = 20;
    break;

case 22:
    // --
    if (*cp == '>')
        mState = 1;
    else
        mState = 20;
    break;
}

if (mParsep - mParseBuf == kParseBufSize - 5) // 5 accounts for possible entity
    at end
        charData();
}
if (final)
    endDocument();
}

```

```

// BLTokenTable.cpp
#include "BLTokenTable.h"

BLTokenTable::BLTokenTable()
: ACE_Hash_Map_Manager<const char*, int, ACE_Null_Mutex>(1009)
{
}

BLTokenTable::~BLTokenTable()
{
    for (iterator i = begin();
         i != end();
         i++)
    {
        char *str = (char*) (*i).ext_id;
        delete[] str;
    }
}

// BLTransformDefs.cpp
#include "BLDocTransformer.h"
#include "BLImageTransformer.h"

BLDocTransformerFactory docTransformer;
BLImageTransformerFactory imgConv;

BLTransformationFactory* BLTransformManager::sFactoryDefs[] =
{
    &docTransformer,
    &imgConv,
    NULL
};

// BLTransformManager.cpp
#include "BLTransformManager.h"
#include "ace/OS.h"
#include "BLTransformation.h"
#include "BLClientInfo.h"

#include "ace/Hash_Map_Manager.h"
#include "ace/Synch.h"

class BLTrivialTransformer : public BLTransformation
{
public:
    BLTrivialTransformer(BLTransformManager::ContentType fromType,
        BLTransformManager::ContentType toType, BLTransportStream *stream)
    : BLTransformation(fromType, toType), mTransportStreamP(stream) {}

    virtual BLTransportStream* getTransportStream()
    {
        return mTransportStreamP;
    }

private:
    BLTransportStream *mTransportStreamP;
};

BLTransformManager* BLTransformManager::sInstanceP = NULL;
BLTransformManager*
BLTransformManager::Instance()
{
    if (sInstanceP == NULL)
        sInstanceP = new BLTransformManager();
    return sInstanceP;
}

```

```

BLTransformManager::BLTransformManager()
{
    ACE_OS::memset(mContentStrings, 0, sizeof(char*) * kNumContentTypes);

    BLTransformManager::BLTransformManager()
    {
    }

    int
    BLTransformManager::initialize()
    {
        ACE_NEW_RETURN(mContentMapP, ContentMap, -1);

        setContentTypeString(kHTMLType, "text/html");
        setContentTypeString(kXMLType, "text/vnd.wap.wml");
        setContentTypeString(kWMLCType, "text/vnd.wap.wmlc");
        setContentTypeString(kJPEGType, "image/jpeg");
        setContentTypeString(kGIFType, "image/gif");
        setContentTypeString(kPNGType, "image/png");
        setContentTypeString(kWMFType, "image/vnd.wap.wbmp");
        setContentTypeString(kPalmPType, "image/vnd.palm.pbm");
        setContentTypeString(kPalmMPType, "application/x-www-form-urlencoded");

        for (BLTransformationFactory* factoryP = sFactoryDefs;
             *factoryP != NULL;
             factoryP++)
        {
            if ((*factoryP)->initialize(this) < 0)
            {
                return -1;
            }
        }

        return 0;
    }

    void
    BLTransformManager::finalize()
    {
        delete mContentMapP;

        for (BLTransformationFactory* factoryP = sFactoryDefs;
             *factoryP != NULL;
             factoryP++)
        {
            (*factoryP)->finalize();
        }
    }

    void
    BLTransformManager::registerTransformation(ContentType fromType,
        ContentType toType,
        BLTransformationFactory *factoryP)
    {
        if (fromType >= 0 && fromType < kNumContentTypes)
        {
            ContentPair pair;
            pair.first = toType;
            pair.second = factoryP;
            mTransformationTable[fromType].push_back(pair);
        }
    }

    BLTransformation*
    BLTransformManager::getTransformation(BLClientInfo *infoP,
        char* fromType,
        BLTransportStream *streamP)
    {
        return getTransformation(infoP,
            stringToContentType(fromType),
            streamP);
    }
}

```



```

BLTransformManager::getTransformation(BLClientInfo *info,
ContentTypeInfo *infoP,
ContentTypeInfo *infoP,
BLTransportStream *streamP)
{
    if (fromType >= 0 && fromType < kNumContentTypes)
    {
        std::vector<ContentTypeInfo> *types = infoP->getAcceptTypes();
        for (std::vector<ContentPair>::iterator i = mTransformationTable[fromType].begin();
             i != mTransformationTable[fromType].end();
             i++)
        {
            for (std::vector<ContentTypeInfo>::iterator j = types->begin();
                 j != types->end();
                 j++)
            {
                if (*j == i->first)
                {
                    return i->second->makeTransformation(fromType, i->first, infoP, streamP);
                }
            }
        }
        for (std::vector<ContentTypeInfo>::iterator j = types->begin();
             j != types->end();
             j++)
        {
            if (*j == fromType)
            {
                return new BLTrivialTransformer(fromType, fromType, streamP);
            }
        }
        return NULL;
    }
    BLTransformManager::ContentTypeInfo
    BLTransformManager::stringToContentType(char *str)
    {
        if (str != NULL)
        {
            ACE_Hash_Map_Entry<const char*, ContentType> *entry;
            char *cp = ACE_OS::strchr(str, ':');
            if (cp == NULL) // no semi-colon
            {
                if (mContentMap->find(str, entry) == 0)
                    return entry->int_id;
            }
            else // semi-colon
            {
                char *typeCopy;
                int len = cp - str;
                ACE_NEW_RETURN(typeCopy,
                               char [len + 1],
                               kUnknownType);
                ACE_OS::strcpy(typeCopy, str, len);
                typeCopy[len] = '\0';
                int err = mContentMap->find(typeCopy, entry);
                delete typeCopy;
                if (err == 0)
                    return entry->int_id;
            }
        }
        return kUnknownType;
    }
    const char*
    BLTransformManager::contentTypeToString(ContentType type)
    {

```

```

    if (type >= 0 && type < kNumContentTypes)
        return mContentStrings[type];
    return NULL;
}

void
BLTransformManager::setContentTypeInfoString(ContentType type, char *str)
{
    mContentStrings[type] = str;
    mContentMap->bind(str, type);
}

// BLWbxmlOutput.cpp
#include "BLWbxmlOutput.h"
#include "BLDocType.h"
BLWbxmlTable::BLWbxmlTable()
: mElementTable(NULL),
  mAttributeTable(NULL),
  mValueTable(NULL)
{
}

BLWbxmlTable::~BLWbxmlTable()
{
    delete[] mElementTable;
    delete[] mAttributeTable;
    delete[] mValueTable;
}

inline unsigned char
BLWbxmlTable::getElementTok(int elemTok)
{
    return mElementTable[elemTok];
}

inline unsigned char
BLWbxmlTable::getDefaultTok(int attrTok)
{
    return mAttributeTable[attrTok].mDefaultTok;
}

inline unsigned char
BLWbxmlTable::getValueTok(int valueTok)
{
    return mValueTable[valueTok];
}

inline BLWbxmlTable::TokVector*
BLWbxmlTable::getAttrToks(int attrTok)
{
    return smAttributeTable[attrTok].mTokVector;
}

BLWbxmlOutput::BLWbxmlOutput(BLTransportStream *transportP, BLWbxmlTable *tableP,
BLTransformManager::ContentType type)
: mStream(transportP),
  mTransportP(transportP),
  mTableP(tableP),
  mContentType(type)
{
}

int
BLWbxmlOutput::beginDoc(BLDocType *docTypeP)
{
    mDocTypeP = docTypeP;
    mTransportP->init(BLTransportStream::kUnknownLength, mContentType);
    mStream << (char) 0x2;
    mStream << (char) 0x8;
    mStream << (char) 0x6A;
    mStream << (char) 0x0;
}

```

```

    return 0;
}

int
BLWbxmlOutput::endDoc()
{
    mStream.flush();
    return 0;
}

int
BLWbxmlOutput::charData(const char* data, int length)
{
    if (!mInCData)
    {
        mInCData = true;
        mStream << '\03';
    }
    mStream.write(data, length);
    return 0;
}

int
BLWbxmlOutput::beginElement(BLXmlElement *elemP)
{
    if (!mInCData)
    {
        mInCData = false;
        mStream << '\0';
    }
    unsigned char tok = mTableP->getElementTok(elemP->getTok());
    if (tok != 0)
    {
        if (!elemP->isEmpty())
            tok |= 0x40;
        BLXmlElement::AttrVector *attrP = elemP->getAttrVector();
        if (attrP->size() > 0)
        {
            tok |= 0x80;
            mStream << tok;
            for (BLXmlElement::AttrVector::iterator i = attrP->begin();
                 i != attrP->end();
                 i++)
            {
                writeAttr(*i);
            }
            mStream << '\01';
        }
        else
        {
            mStream << tok;
        }
    }
    return 0;
}

int
BLWbxmlOutput::endElement(int /*elemTok*/)
{
    if (!mInCData)
    {
        mInCData = false;
        mStream << '\0';
    }
    mStream << '\01';
    return 0;
}

void
BLWbxmlOutput::writeAttr(BLAttribute *attrP)
{
    BLWbxmlTable::TokVector *tokSP = mTableP->getAttrToks(attrP->getTok());
    for (BLWbxmlTable::TokVector::iterator i = tokSP->begin();
         i != tokSP->end();
         i++)
    {
        int index;
        if (i->second.getType() == attrP->getType())
        {
            switch (attrP->getType())
            {
                case BLAttribute::kCDataType:
                {
                    char *defCData = i->second.getCDataValue();
                    char *attrCData = attrP->getCDataValue();
                    index = 0;
                    while (true)
                    {
                        if (defCData[index] == '\0')
                        {
                            mStream << i->first;
                            if (attrCData[index] != '\0')
                            {
                                mStream << '\03';
                                mStream << (attrCData + index);
                                mStream << '\0';
                            }
                            return;
                        }
                        if (defCData[index] != attrCData[index])
                        {
                            break;
                        }
                        index++;
                    }
                }
                break;
                case BLAttribute::kTokenType:
                {
                    if (attrP->getTokValue() == i->second.getTokValue())
                    {
                        mStream << i->first;
                        return;
                    }
                }
                break;
                case BLAttribute::kBoolType:
                {
                    if (attrP->getBoolValue() == i->second.getBoolValue())
                    {
                        mStream << i->first;
                        return;
                    }
                }
                break;
            }
        }
    }
    int defaultTok = mTableP->getDefaultTok(attrP->getTok());
    if (defaultTok != 0)
    {
        switch (attrP->getType())
        {
            case BLAttribute::kCDataType:
            {
                mStream << defaultTok;
                mStream << '\03';
                mStream << attrP->getCDataValue();
                mStream << '\0';
            }
            break;
            case BLAttribute::kIntType:
            {
                mStream << defaultTok;
                mStream << '\03';
                mStream << attrP->getIntValue();
                mStream << '\0';
            }
            break;
            case BLAttribute::kLengthType:
            {
                mStream << defaultTok;
                mStream << '\03';
            }
            int intValue = attrP->getIntValue();
        }
    }
}

```

```
if (intValue >= 0)
    mStream << intValue;
else
    mStream << -intValue << '%';
}

mStream << '\0';
break;
case BLAttribute::kTokenType:
{
    mStream << defaultTok;

    unsigned char valueTok = mTableP->getValueTok(attrP->getTokenValue());
    if (valueTok != 0)
    {
        mStream << valueTok;
    }
    else
    {
        mStream << '\03';
        mStream << mDocTypeP->getValueString(attrP->getTokenValue());
        mStream << '\0';
    }
    break;
}
}

// BLWmlDocType.cpp
#include "BLWmlDocType.h"

// element classes
enum {
    kEmphClass      = 01,
    kLayoutClass    = 02,
    kTextClass      = 04,
    kFlowClass      = 010,
    kTaskClass      = 020,
    kNavElmtsClass  = 040,
    kFieldsClass    = 0100,
};

BLWmlDocType::BLWmlDocType()
: BLDocType(kNumElements)
{
}

int
BLWmlDocType::initialize()
{
    if (initTables(kNumElements, kNumAttrs, kNumValues) < 0)
        return -1;

    initClasses();
    initElements();
    if (initPaths() < 0)
        return -1;

    initElementToks();
    initAttributeToks();
    initValueToks();
    initEntityToks();

    return 0;
}

void
BLWmlDocType::initClasses()
{
    // emph
    mElementDefs[kEmphasisElement] .addToClass(kEmphClass);
    mElementDefs[kStrongElement]    .addToClass(kEmphClass);
    mElementDefs[kBoldElement]      .addToClass(kEmphClass);
    mElementDefs[kItalicElement]    .addToClass(kEmphClass);
    mElementDefs[kUnderlineElement].addToClass(kEmphClass);
}
```

```
mElementDefs[kBigElement]      .addToClass(kEmphClass);
mElementDefs[kSmallElement]    .addToClass(kEmphClass);
// layout
mElementDefs[kLinebreakElement].addToClass(kLayoutClass);

// text
mElementDefs[kCDATAElement]    .addToClass(kTextClass);
addClassToClass(kTextClass, kEmphClass);

// flow
mElementDefs[kImageElement]    .addToClass(kFlowClass);
mElementDefs[kAnchorElement]   .addToClass(kFlowClass);
mElementDefs[kAElement]        .addToClass(kFlowClass);
mElementDefs[kTableElement]    .addToClass(kFlowClass);
addClassToClass(kFlowClass, kTextClass);
addClassToClass(kFlowClass, kLayoutClass);

// task
mElementDefs[kGoElement]       .addToClass(kTaskClass);
mElementDefs[kRevElement]      .addToClass(kTaskClass);
mElementDefs[kNoopElement]     .addToClass(kTaskClass);
mElementDefs[kRefreshElement]  .addToClass(kTaskClass);

// navElmts
mElementDefs[kDoElement]       .addToClass(kNavElmtsClass);
mElementDefs[kOnEventElement]  .addToClass(kNavElmtsClass);

// fields
mElementDefs[kInputElement]    .addToClass(kFieldsClass);
mElementDefs[kSelectElement]   .addToClass(kFieldsClass);
mElementDefs[kFieldSetElement] .addToClass(kFieldsClass);
addClassToClass(kFieldsClass, kFlowClass);

void
BLWmlDocType::initElements()
{
    // attributes
    BLAttributeDef *cdDataNameImp
        = makeAttrDef(kNameAttr, BLAttribute::kCDATAtype,
        false);
    BLAttributeDef *cdDataNameReq
        = makeAttrDef(kNameAttr, BLAttribute::kCDATAtype,
        true);
    BLAttributeDef *title
        = makeAttrDef(kTitleAttr, BLAttribute::kCDATAtype,
        false);
    BLAttributeDef *newContext
        = makeAttrDef(kNewContextAttr,
        BLAttribute::kBooleanType, false);
    BLAttributeDef *ordered
        = makeAttrDef(kOrderedAttr, BLAttribute::kBooleanType,
        false);
    BLAttributeDef *onEnterForward = makeAttrDef(kOnEnterForwardAttr,
    BLAttribute::kCDATAtype, false);
    BLAttributeDef *onEnterBackward = makeAttrDef(kOnEnterBackwardAttr,
    BLAttribute::kCDATAtype, false);
    BLAttributeDef *onTimer
        = makeAttrDef(kOnTimerAttr, BLAttribute::kCDATAtype,
        false);
    BLAttributeDef *label
        = makeAttrDef(kLabelAttr, BLAttribute::kCDATAtype,
        false);
    BLAttributeDef *optional
        = makeAttrDef(kOptionalAttr, BLAttribute::kBooleanType,
        false);
    BLAttributeDef *type
        = makeAttrDef(kTypeAttr, BLAttribute::kCDATAtype,
        true);
    BLAttributeDef *doType
        = makeAttrDef(kTypeAttr, BLAttribute::kTokenType,
        true);
    doType->addValueTok(kAcceptValue);
    doType->addValueTok(kPrevValue);
    doType->addValueTok(kHelpValue);
    doType->addValueTok(kResetValue);
    doType->addValueTok(kOptionsValue);
    doType->addValueTok(kDeleteValue);
    doType->addValueTok(kUnknownValue);
    BLAttributeDef *oneventType = makeAttrDef(kAcceptValue);
    oneventType->addValueTok(kAcceptValue);
    oneventType->addValueTok(kDeleteValue);
    oneventType->addValueTok(kHelpValue);
    oneventType->addValueTok(kPasswordValue);
}
```





```

addElementTok("select", kSelectElement);
addElementTok("optgroup", kOptgroupElement);
addElementTok("option", kOptionElement);
addElementTok("input", kInputElement);
addElementTok("fieldset", kFieldSetElement);
addElementTok("div", kDivElement);
addElementTok("img", kImageElement);
addElementTok("table", kTableElement);
addElementTok("tr", kTableRowElement);
addElementTok("td", kTableDataElement);
addElementTok("em", kEmphasisElement);
addElementTok("strong", kStrongElement);
addElementTok("b", kBoldElement);
addElementTok("i", kItalicElement);
addElementTok("u", kUnderlineElement);
addElementTok("small", kSmallElement);
addElementTok("big", kBigElement);
addElementTok("p", kParagraphElement);
addElementTok("br", kLinebreakElement);
addElementTok("pre", kPreElement);
}

void
BLWmlDocType::initAttributeToks()
{
    addAttributeTok("href", kHrefAttr);
    addAttributeTok("name", kNameAttr);
    addAttributeTok("value", kValueAttr);
    addAttributeTok("align", kAlignAttr);
    addAttributeTok("src", kSrcAttr);
    addAttributeTok("localsrc", kLocalSrcAttr);
    addAttributeTok("alt", kAltAttr);
    addAttributeTok("vspace", kVSpaceAttr);
    addAttributeTok("hspace", kHSpaceAttr);
    addAttributeTok("height", kHeightAttr);
    addAttributeTok("width", kWidthAttr);
    addAttributeTok("title", kTitleAttr);
    addAttributeTok("newcontext", kNewContextAttr);
    addAttributeTok("ordered", kOrderedAttr);
    addAttributeTok("onenterforward", kOnEnterForwardAttr);
    addAttributeTok("onenterbackward", kOnEnterBackwardAttr);
    addAttributeTok("ontimer", kTimerAttr);
    addAttributeTok("type", kTypeAttr);
    addAttributeTok("domain", kDomainAttr);
    addAttributeTok("path", kPathAttr);
    addAttributeTok("http-equiv", kHttpEquivAttr);
    addAttributeTok("forua", kForuaAttr);
    addAttributeTok("content", kContentAttr);
    addAttributeTok("scheme", kSchemeAttr);
    addAttributeTok("sendreferer", kSendRefererAttr);
    addAttributeTok("method", kMethodAttr);
    addAttributeTok(" enctype", kEnctypeAttr);
    addAttributeTok("accept-charset", kAcceptCharsetAttr);
    addAttributeTok("iname", kInameAttr);
    addAttributeTok("ivalue", kIValueAttr);
    addAttributeTok("multiple", kMultipleAttr);
    addAttributeTok("tabindex", kTabIndexAttr);
    addAttributeTok("onpick", kOnPickAttr);
    addAttributeTok("format", kFormatAttr);
    addAttributeTok("emptyok", kEmptyOkAttr);
    addAttributeTok("size", kSizeAttr);
    addAttributeTok("maxlength", kMaxLengthAttr);
    addAttributeTok("accesskey", kAccessKeyAttr);
    addAttributeTok("label", kLabelAttr);
    addAttributeTok("optional", kOptionalAttr);
    addAttributeTok("mode", kModeAttr);
    addAttributeTok("id", kIDAttr);
}

void
BLWmlDocType::initValueToks()

```

```

addValueTok("left", kLeftValue);
addValueTok("center", kCenterValue);
addValueTok("right", kRightValue);
addValueTok("top", kTopValue);
addValueTok("middle", kMiddleValue);
addValueTok("bottom", kBottomValue);
addValueTok("accept", kAcceptValue);
addValueTok("delete", kDeleteValue);
addValueTok("help", kHelpValue);
addValueTok("password", kPasswordValue);
addValueTok("onpick", kOnPickValue);
addValueTok("onenterbackward", kOnEnterBackwardValue);
addValueTok("onenterforward", kOnEnterForwardValue);
addValueTok("ontimer", kTimerValue);
addValueTok("options", kOptionsValue);
addValueTok("prev", kPrevValue);
addValueTok("reset", kResetValue);
addValueTok("text", kTextValue);
addValueTok("unknown", kUnknownValue);
}

void
BLWmlDocType::initEntityToks()
{
    addEntityTok("quot", 34);
    addEntityTok("amp", 38);
    addEntityTok("apos", 39);
    addEntityTok("lt", 60);
    addEntityTok("gt", 62);
    addEntityTok("nbsp", 160);
    addEntityTok("shy", 173);
}

const char*
BLWmlDocType::getDocTypeString()
{
    return "<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"  

    \"http://www.wapforum.org/DTD/wml_1.1.xml\">";
}

BLElementDef*
BLWmlDocType::getRootElemDef()
{
    return getElementDef(kWmlElement);
}

// BLWmlTable.cpp
#include "BLWmlTable.h"
#include "BLWmlDocType.h"

int
BLWmlTable::initialize()
{
    if (makeElementTable() < 0 ||  

        makeAttributeTable() < 0)  

        return -1;

    return 0;
}

int
BLWmlTable::makeElementTable()
{
    ACE_NEW_RETURN(mElementTable,  

        unsigned char[BLWmlDocType::kNumElements],  

        -1);

    ACE_OS::memset(mElementTable, 0, BLWmlDocType::kNumElements * sizeof(unsigned char));

    mElementTable[BLWmlDocType::kWmlElement]  

        = 0x3F;  

    mElementTable[BLWmlDocType::kCardElement]  

        = 0x27;  

    mElementTable[BLWmlDocType::kDoElement]  

        = 0x26;  

    mElementTable[BLWmlDocType::kCommandEvent]  

        = 0x33;  

    mElementTable[BLWmlDocType::kHeadElement]  

        = 0x3B;
}

```

```

mElementTable[BLMmlDocType::kAccessElement] = 0x23;
mElementTable[BLMmlDocType::kMetaElement] = 0x30;
mElementTable[BLMmlDocType::kPageHeader] = 0x28;
mElementTable[BLMmlDocType::kPageFooter] = 0x32;
mElementTable[BLMmlDocType::kPageHeader] = 0x36;
mElementTable[BLMmlDocType::kPageFooter] = 0x31;
mElementTable[BLMmlDocType::kPageHeader] = 0x21;
mElementTable[BLMmlDocType::kPageFooter] = 0x35;
mElementTable[BLMmlDocType::kPageHeader] = 0x37;
mElementTable[BLMmlDocType::kPageFooter] = 0x33;
mElementTable[BLMmlDocType::kPageHeader] = 0x2F;
mElementTable[BLMmlDocType::kPageFooter] = 0x2A;
mElementTable[BLMmlDocType::kPageHeader] = 0x3C;
mElementTable[BLMmlDocType::kPageFooter] = 0x2E;
mElementTable[BLMmlDocType::kPageHeader] = 0x22;
mElementTable[BLMmlDocType::kPageFooter] = 0x1C;
mElementTable[BLMmlDocType::kPageHeader] = 0x1E;
mElementTable[BLMmlDocType::kPageFooter] = 0x1B;
mElementTable[BLMmlDocType::kPageHeader] = 0x1D;
mElementTable[BLMmlDocType::kPageFooter] = 0x26;
mElementTable[BLMmlDocType::kPageHeader] = 0x1B;
mElementTable[BLMmlDocType::kPageFooter] = 0x1B;

return 0;

int
BLMmlTable::makeValueTable()
{
    ACE_NEW_RETURN(mValueTable,
        unsigned char[BLMmlDocType::KNumValues],
        -1);
    ACE_OS::memset(mValueTable, 0, BLMmlDocType::KNumValues * sizeof(unsigned char));

    mValueTable[BLMmlDocType::kTopValue] = 0x9E;
    mValueTable[BLMmlDocType::kMiddleValue] = 0x93;
    mValueTable[BLMmlDocType::kBottomValue] = 0x8A;
    mValueTable[BLMmlDocType::kWrapValue] = 0xA0;
    mValueTable[BLMmlDocType::kNoWrapValue] = 0x94;
    mValueTable[BLMmlDocType::kAcceptValue] = 0x89;
    mValueTable[BLMmlDocType::kDeleteValue] = 0x8C;
    mValueTable[BLMmlDocType::kHelpValue] = 0x8D;
    mValueTable[BLMmlDocType::kPasswordValue] = 0x9A;
    mValueTable[BLMmlDocType::kPickValue] = 0x95;
    mValueTable[BLMmlDocType::kOnEnterBackwardValue] = 0x96;
    mValueTable[BLMmlDocType::kOnEnterForwardValue] = 0x97;
    mValueTable[BLMmlDocType::kOptionsValue] = 0x98;
    mValueTable[BLMmlDocType::kOptionsValue] = 0x99;
    mValueTable[BLMmlDocType::kResetValue] = 0x9B;
    mValueTable[BLMmlDocType::kResetValue] = 0x9D;
    mValueTable[BLMmlDocType::kUnknownValue] = 0x9F;

    return 0;
}

int
BLMmlTable::makeAttributeTable()
{
    ACE_NEW_RETURN(mAttributeTable,
        Attribute[BLMmlDocType::KNumAttrs],
        -1);

    BLAttribute attr;

    // href
    mAttributeTable[BLMmlDocType::kHrefAttr].mDefaultTok = 0x4A;
    attr.setType(BLAttribute::kCDATAtype);

```

```

attr.setCdataValue("http://");
mAttributeTable[BLMmlDocType::kHrefAttr].mTokVector.push(attr, 0x4B);
attr.setCdataValue("https://");
mAttributeTable[BLMmlDocType::kHrefAttr].mTokVector.push(attr, 0x4C);
// name
mAttributeTable[BLMmlDocType::kNameAttr].mDefaultTok = 0x21;
// value
mAttributeTable[BLMmlDocType::kNameAttr].mDefaultTok = 0x4D;
// align
mAttributeTable[BLMmlDocType::kAlignAttr].mDefaultTok = 0x52;
attr.setType(BLAttribute::kTokenType);
mAttributeTable[BLMmlDocType::kBottomValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0x6);
mAttributeTable[BLMmlDocType::kCenterValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0x7);
mAttributeTable[BLMmlDocType::kLeftValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0x8);
mAttributeTable[BLMmlDocType::kMiddleValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0x9);
mAttributeTable[BLMmlDocType::kRightValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0xA);
mAttributeTable[BLMmlDocType::kTopValue];
mAttributeTable[BLMmlDocType::kAlignAttr].mTokVector.push(attr, 0xB);
// src
mAttributeTable[BLMmlDocType::kSrcAttr].mDefaultTok = 0x32;
attr.setType(BLAttribute::kCDATAtype);
attr.setCdataValue("http://");
mAttributeTable[BLMmlDocType::kSrcAttr].mTokVector.push(attr, 0x58);
attr.setCdataValue("https://");
mAttributeTable[BLMmlDocType::kSrcAttr].mTokVector.push(attr, 0x59);
// localsrc
mAttributeTable[BLMmlDocType::kLocalSrcAttr].mDefaultTok = 0x19;
// alt
mAttributeTable[BLMmlDocType::kAltAttr].mDefaultTok = 0xC;
// vspace
mAttributeTable[BLMmlDocType::kVSpaceAttr].mDefaultTok = 0x4E;
// hspace
mAttributeTable[BLMmlDocType::kHSpaceAttr].mDefaultTok = 0x14;
// height
mAttributeTable[BLMmlDocType::kHeightAttr].mDefaultTok = 0x13;
// width
mAttributeTable[BLMmlDocType::kWidthAttr].mDefaultTok = 0x4F;
// title
mAttributeTable[BLMmlDocType::kTitleAttr].mDefaultTok = 0x36;
// newcontext
mAttributeTable[BLMmlDocType::kNewContextAttr].mDefaultTok = 0x0;
attr.setType(BLAttribute::kTokenType);
attr.setBoolValue(false);
mAttributeTable[BLMmlDocType::kNewContextAttr].mTokVector.push(attr, 0x28);
attr.setBoolValue(true);
mAttributeTable[BLMmlDocType::kNewContextAttr].mTokVector.push(attr, 0x29);
// ordered
mAttributeTable[BLMmlDocType::kOrderedAttr].mDefaultTok = 0x0;
attr.setType(BLAttribute::kTokenType);
attr.setBoolValue(false);
mAttributeTable[BLMmlDocType::kOrderedAttr].mTokVector.push(attr, 0x28);
attr.setBoolValue(true);
mAttributeTable[BLMmlDocType::kOrderedAttr].mTokVector.push(attr, 0x29);
// onenterforward
mAttributeTable[BLMmlDocType::kOnEnterForwardAttr].mDefaultTok = 0x26;
// onenterbackward
mAttributeTable[BLMmlDocType::kOnEnterBackwardAttr].mDefaultTok = 0x25;
// ontimer
mAttributeTable[BLMmlDocType::kOnTimerAttr].mDefaultTok = 0x27;
// type
mAttributeTable[BLMmlDocType::kTypeAttr].mDefaultTok = 0x37;
attr.setType(BLAttribute::kTokenType);
mAttributeTable[BLMmlDocType::kAcceptValue];
mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(attr, 0x38);
mAttributeTable[BLMmlDocType::kDeleteValue];
mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(attr, 0x39);
mAttributeTable[BLMmlDocType::kHelpValue];
mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(attr, 0x2A);
mAttributeTable[BLMmlDocType::kPasswordValue];
mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(attr, 0x3B);

```

```

    attr.setTokValue(BLMmlDocType::kOnPickValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x3C);
    attr.setTokValue(BLMmlDocType::kOnEnterBackwardValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x3D);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x3E);
    attr.setTokValue(BLMmlDocType::kOnEnterForwardValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x3F);
    attr.setTokValue(BLMmlDocType::kOnEnterValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x40);
    attr.setTokValue(BLMmlDocType::kOptionsValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x41);
    attr.setTokValue(BLMmlDocType::kPrevValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x42);
    attr.setTokValue(BLMmlDocType::kResetValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x43);
    attr.setTokValue(BLMmlDocType::kResetValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x44);
    attr.setTokValue(BLMmlDocType::kTextValue);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x45);
    mAttributeTable[BLMmlDocType::kTypeAttr].mTokVector.push(&attr, 0x46);
    // domain
    mAttributeTable[BLMmlDocType::kDomainAttr].mDefaultTok = 0xF;
    // path
    mAttributeTable[BLMmlDocType::kPathAttr].mDefaultTok = 0x2A;
    // http-equiv
    mAttributeTable[BLMmlDocType::kHttpequivAttr].mDefaultTok = 0x5A; // ignore
    // forua
    mAttributeTable[BLMmlDocType::kForuaAttr].mDefaultTok = 0x12;
    // content
    mAttributeTable[BLMmlDocType::kContentAttr].mDefaultTok = 0xD;
    attr.setType(BLAAttribute::kContentType);
    attr.setCdataValue("application/vnd.wap.xml; charset=");
    mAttributeTable[BLMmlDocType::kContentAttr].mTokVector.push(&attr, 0x5C);
    // scheme
    mAttributeTable[BLMmlDocType::kSchemeAttr].mDefaultTok = 0x2E;
    // send-referer
    mAttributeTable[BLMmlDocType::kSendRefererAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kContentType);
    attr.setBoolValue(false);
    mAttributeTable[BLMmlDocType::kSendRefererAttr].mTokVector.push(&attr, 0x2F);
    attr.setBoolValue(true);
    mAttributeTable[BLMmlDocType::kSendRefererAttr].mTokVector.push(&attr, 0x30);
    // method
    mAttributeTable[BLMmlDocType::kMethodAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kTokenType);
    attr.setTokValue(BLMmlDocType::kGetValue);
    mAttributeTable[BLMmlDocType::kMethodAttr].mTokVector.push(&attr, 0x1B);
    attr.setTokValue(BLMmlDocType::kMethodAttr);
    mAttributeTable[BLMmlDocType::kMethodAttr].mTokVector.push(&attr, 0x1C);
    // enctype
    mAttributeTable[BLMmlDocType::kEnctypeAttr].mDefaultTok = 0x5F;
    attr.setType(BLAAttribute::kContentType);
    attr.setCdataValue("application/x-www-form-urlencoded");
    mAttributeTable[BLMmlDocType::kEnctypeAttr].mTokVector.push(&attr, 0x60);
    attr.setCdataValue("multipart/form-data");
    mAttributeTable[BLMmlDocType::kEnctypeAttr].mTokVector.push(&attr, 0x61);
    // accept-charset
    mAttributeTable[BLMmlDocType::kAcceptCharsetAttr].mDefaultTok = 0x5;
    // iname
    mAttributeTable[BLMmlDocType::kNameAttr].mDefaultTok = 0x16;
    // ivalue
    mAttributeTable[BLMmlDocType::kValueAttr].mDefaultTok = 0x15;
    // multiple
    mAttributeTable[BLMmlDocType::kMultipleAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kBoolType);
    attr.setBoolValue(false);
    mAttributeTable[BLMmlDocType::kMultipleAttr].mTokVector.push(&attr, 0x1F);
    mAttributeTable[BLMmlDocType::kMultipleAttr].mTokVector.push(&attr, 0x20);
    // tabIndex
    mAttributeTable[BLMmlDocType::kTabIndexAttr].mDefaultTok = 0x35;
    // onpick
    mAttributeTable[BLMmlDocType::kOnPickAttr].mDefaultTok = 0x24;
    // format
    mAttributeTable[BLMmlDocType::kFormatAttr].mDefaultTok = 0x12;
    // emptytok
    mAttributeTable[BLMmlDocType::kEmptyTokAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kBoolType);
    attr.setBoolValue(false);

```

```

    mAttributeTable[BLMmlDocType::kEmptyTokAttr].mTokVector.push(&attr, 0x10);
    attr.setBoolValue(true);
    mAttributeTable[BLMmlDocType::kEmptyTokAttr].mTokVector.push(&attr, 0x11);
    // size
    mAttributeTable[BLMmlDocType::kSizeAttr].mDefaultTok = 0x31;
    // maxLength
    mAttributeTable[BLMmlDocType::kMaxLengthAttr].mDefaultTok = 0x1A;
    // accesskey
    mAttributeTable[BLMmlDocType::kAccessKeyAttr].mDefaultTok = 0x5E;
    // columns
    mAttributeTable[BLMmlDocType::kColumnsAttr].mDefaultTok = 0x53;
    // label
    mAttributeTable[BLMmlDocType::kLabelAttr].mDefaultTok = 0x16;
    // optional
    mAttributeTable[BLMmlDocType::kOptionalAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kBoolType);
    attr.setBoolValue(false);
    mAttributeTable[BLMmlDocType::kOptionalAttr].mTokVector.push(&attr, 0x28);
    attr.setBoolValue(true);
    mAttributeTable[BLMmlDocType::kOptionalAttr].mTokVector.push(&attr, 0x29);
    // mode
    mAttributeTable[BLMmlDocType::kModeAttr].mDefaultTok = 0x0;
    attr.setType(BLAAttribute::kTokenType);
    attr.setTokValue(BLMmlDocType::kWrapValue);
    mAttributeTable[BLMmlDocType::kModeAttr].mTokVector.push(&attr, 0x1D);
    attr.setTokValue(BLMmlDocType::kWrapValue);
    mAttributeTable[BLMmlDocType::kModeAttr].mTokVector.push(&attr, 0x1E);
    // id
    mAttributeTable[BLMmlDocType::kIDAttr].mDefaultTok = 0x55;
    return 0;
} // BLMmlOutput.cpp
#include "BLXmlOutput.h"
#include "BLDocType.h"
BLXmlOutput::BLXmlOutput(BLTransportStream *transportP, BLTransformManager::ContentType
type)
: mStream(transportP),
  mTransportP(transportP),
  mDocTypeP(NULL),
  mContentType(type)
{
}
inline void
BLXmlOutput::writeChar(char c)
{
    unsigned int intVal = c;
    if (intVal < 0x20)
    {
        if (c == '\r' || c == '\n')
            mStream << c;
        else if (intVal > 0x7E)
            mStream << "&#" << intVal << ";";
        else
        {
            switch (c)
            {
                case '"':
                    mStream << "&quot;";
                    break;
                case '&':
                    mStream << "&amp;";
                    break;
                case '\':
                    mStream << "&apos;";
                    break;
                case '<':
                    mStream << "&lt;";
                    break;
                case '>':
                    mStream << "&gt;";
                    break;
            }
        }
    }
}

```



```

        mStream << "gt;";
        break;
    default:
        mStream << c;
        break;
    }
}

inline int
BLXmlOutput::charData(const char* data, int length)
{
    for (int i=0; i<length; i++)
        writeChar(data[i]);
    return 0;
}

inline void
BLXmlOutput::charData(const char* str)
{
    while (*str != '\0')
        writeChar(*str++);
}

int
BLXmlOutput::beginDoc(BLDocType* docTypeP)
{
    mDocTypeP = docTypeP;
    mTransportP->init(BLTransportStream::UnknownLength, mContentType);
    mStream << "<?xml version='1.0'>";
    mStream << docTypeP->getDocTypeString();
    return 0;
}

int
BLXmlOutput::endDoc()
{
    mStream.flush();
    return 0;
}

int
BLXmlOutput::beginElement(BLElement* elemP)
{
    mStream << '<' << mDocTypeP->getElementString(elemP->getTok());
    BLElement::AttrVector* attrsp = elemP->getAttrVector();
    for (BLElement::AttrVector::iterator i = attrsp->begin();
         i != attrsp->end();
         i++)
    {
        mStream << ' ' << mDocTypeP->getAttributeString(i->getTok()) << "=" << "\"";
        switch (i->getType())
        {
            case BLAttribute::kTokenType:
                mStream << mDocTypeP->getValueString(i->getValue());
                break;
            case BLAttribute::kIntType:
                mStream << i->getIntValue();
                break;
            case BLAttribute::kCDataType:
                charData(i->getCDataValue());
                break;
            case BLAttribute::kBoolType:
                mStream << i->getBoolValue() ? "true" : "false";
                break;
            case BLAttribute::kLengthType:
                int intValue = i->getIntValue();
                if (intValue >= 0)
                    mStream << intValue;
                else
                    mStream << "-intValue << '";

```

```

        break;
    case BLAttribute::kNoValueType:
        mStream << mDocTypeP->getAttributeString(i->getTok());
        break;
    }
    mStream << '\n';
    if (elemP->isEmpty())
        mStream << '/';
    mStream << '>';
    return 0;
}

int
BLXmlOutput::endElement(int elemTok)
{
    mStream << "</" << mDocTypeP->getElementString(elemTok) << '>';
    return 0;
}

// BLXmlParser.cpp
#include "BLXmlParser.h"
#define kParseBufSize 2048

BLXmlParser::BLXmlParser(BLDocType* docTypeP, BLDocStream* docStreamP)
: mDocTypeP(docTypeP),
  mDocStreamP(docStreamP),
  mLastWhitespace(true),
  mPreserveWhitespace(false),
  mPastInitialWhitespace(false),
  mIgnoreWhitespace(false),
  mParser(NULL)
{
}

BLXmlParser::~BLXmlParser()
{
}

int
BLXmlParser::init(int /*length*/, BLTransformManager::ContentType /*type*/)
{
    mParser = XML_ParserCreate(NULL);
    if (mParser == NULL)
        return -1;

    XML_SetUserData(mParser, this);
    XML_SetElementHandler(mParser,
        xmlStartElementHandler,
        xmlEndElementHandler);
    XML_SetCharacterDataHandler(mParser,
        xmlCDataHandler);

    mLastWhitespace = true;
    mPreserveWhitespace = false;
    mPastInitialWhitespace = false;
    mIgnoreWhitespace = false;

    return mDocStreamP->beginDoc(mDocTypeP);
}

int
BLXmlParser::put(char* data, int size, bool final)
{
    // skip initial whitespace (to keep expat from crashing out)
    if (!mPastInitialWhitespace)
    {
        for (int i=0; i<size; i++)

```

```

    if (*data <= 0x20)
        data++;
    else
    {
        mPastInitialWhitespace = true;
        break;
    }
}

if(!XML_Parse(mParser, data, size, final))
{
    error(XML_ErrorString(XML_GetErrorCode(mParser)), XML_GetErrorLineNumber(mParser));
    mDocStreamP->endDoc();
    XML_ParserFree(mParser);
    mParser = NULL;
    return -1;
}

if (final)
{
    mDocStreamP->endDoc();
    XML_ParserFree(mParser);
}

return 0;
}

void
BLXmlParser::xmlStartElementHandler(void *userData,
    const char *name,
    const char **atts)
{
    BLXmlParser *parserP = (BLXmlParser*) userData;

    int elemTok = parserP->mDocTypeP->getElementTok((char*)name);
    if (elemTok != -1)
    {
        BLXmlElement;
        BLAttribute attr;
        element.setTok(elemTok);

        for (const char **attP=atts; *attP!=0; attP++)
        {
            char *attName = (char*)*attP;
            char *attValue = (char*)(**attP);

            int attrTok = parserP->mDocTypeP->getAttributeTok(attName);
            if (attrTok != -1)
            {
                attr.setTok(attrTok);
                attr.setCDataValue(attValue);
                element.addAttribute(attr);
            }
        }

        parserP->mDocStreamP->beginElement(element);

        BLXmlElementDef *defP = parserP->mDocTypeP->getElementDef(elemTok);
        parserP->mPreserveWhitespace = defP->getPreserveWhitespace();
        parserP->mIgnoreWhitespace = defP->getIgnoreWhitespace();
        if (defP->getResetWhitespace()
            parserP->mLastWhitespace = true;

        parserP->mElementStack.push_back(defP);
    }
}

void
BLXmlParser::xmlEndElementHandler(void *userData,
    const char *name)
{
    BLXmlParser *parserP = (BLXmlParser*) userData;

```

```

    int tag = parserP->mDocTypeP->getElementTok((char*)name);
    if (tag != -1)
    {
        parserP->mDocStreamP->endElement(tag);
        parserP->mElementStack.pop_back();
    }
    if (parserP->mElementStack.size() > 0)
    {
        BLXmlElementDef *defP = parserP->mElementStack.back();
        parserP->mPreserveWhitespace = defP->getPreserveWhitespace();
        parserP->mIgnoreWhitespace = defP->getIgnoreWhitespace();
    }

    void
    BLXmlParser::xmlCDataHandler(void *userData,
        const XML_Char *str,
        int len)
    {
        BLXmlParser *parserP = (BLXmlParser*) userData;
        char buf[kParseBufSize];
        int strIndex = 0;
        int bufIndex = 0;

        if (parserP->mPreserveWhitespace)
        {
            parserP->mDocStreamP->charData(str, len);
        }
        else
        {
            while (strIndex < len)
            {
                if (str[strIndex] <= 0x20)
                {
                    if (!parserP->mLastWhitespace && !parserP->mIgnoreWhitespace)
                    {
                        parserP->mLastWhitespace = true;
                        buf[bufIndex++] = ' ';
                    }
                }
                else
                {
                    parserP->mLastWhitespace = false;
                    buf[bufIndex++] = str[strIndex];
                }
                if (bufIndex == kParseBufSize)
                {
                    parserP->mDocStreamP->charData(buf, bufIndex);
                    bufIndex = 0;
                }
                strIndex++;
            }
            if (bufIndex > 0)
            {
                parserP->mDocStreamP->charData(buf, bufIndex);
            }
        }

        void
        BLXmlParser::error(const char *message, int lineNumber)
        {
            char buf[kParseBufSize];
            ACE_OS::sprintf(buf, "XML Parse Error: r\\line %d: %s", lineNumber, message);
            mDocStreamP->charData(buf, ACE_OS::strlen(buf));
        }
        // BLDocNormalizer.cpp

        #include "BLDocNormalizer.h"

        #ifdef ACE_WIN32
        #pragma warning(disable:4786)
        #endif
    }
}

```

```

BLDocNormalizer::BLDocNormalizer(BLDocStream *streamP)
: mDocStreamP(streamP),
  mDocTypeP(NULL),
  mPending(false)
{
}

int
BLDocNormalizer::beginDoc(BLDocType *docTypeP)
{
    mDocTypeP = docTypeP;
    mPending = false;
    mContextVector.push_back(new ElemVector());
    return mDocStreamP->beginDoc(docTypeP);
}

int
BLDocNormalizer::endDoc()
{
    // close all elements
    while (mContextVector.size() > 0)
    {
        ElemVector *elemsP = mContextVector.back();
        while (elemsP->size() > 0)
        {
            if (mDocStreamP->endElement(elemsP->back()->getTok()) < 0)
                return -1;
            elemsP->pop_back();
        }
        delete elemsP;
        mContextVector.pop_back();
    }
    return mDocStreamP->endDoc();
}

int
BLDocNormalizer::charData(const char* data, int length)
{
    if (mContextVector.size() == 0)
        return -1;
    BLElementDef *defP = mDocTypeP->getElementDef(BLDocType::KCDATAElement);
    if (mPending &&

```

```

    topElemP = topElemP->getShortestPath(BLDocType::KCDATAElement);
    return mDocStreamP->charData(data, length);
}

int
BLDocNormalizer::beginElement(BLElement *elemP)
{
    if (mContextVector.size() == 0)
        return -1;
    BLElementDef *defP = mDocTypeP->getElementDef(elemP->getTok());
    if (!defP->validateElement(elemP))
        return 0;
    if (mPending)
    {
        if (elemP->getTok() == mPendingElement.getTok())
        {
            mPendingElement = *elemP;
            return 0;
        }
        else
        {
            if
            (
                (mDocTypeP->getElementDef(mPendingElement.getTok())->getShortestPath(elemP->getTok()) !=
                NULL)
                ||
                (if (openElement(mPendingElement) < 0)
                 return -1;
                 mPending = false;
                )
            )
            {
                if (!defP->getEmptyOK())
                {
                    mPendingElement = *elemP;
                    mPending = true;
                    return 0;
                }
                return openElement(elemP);
            }
        }
    }
    int
    BLDocNormalizer::openElement(BLElement *elemP)
    {
        ElemVector *curContextP = mContextVector.back();
        // check if element already open in this context, if so close it
        for (ElemVector::reverse_iterator ri = curContextP->rbegin();
             ri != curContextP->rend();
             ri++)
        {
            if ((*ri)->getTok() == elemP->getTok())
            {
                return reopen(elemP);
            }
        }
        // handle root case
        if (curContextP->size() == 0)
        {
            BLElementDef *rootP = mDocTypeP->getRootElemDef();
            curContextP->push_back(rootP);
            mElement.setTok(rootP->getTok());
            if (mDocStreamP->beginElement(mElement) < 0)
                return -1;
            if (elemP->getTok() == rootP->getTok())
                return 0;
        }
    }

```

```

        return openElements();
    }
    return 0;
}

int
BLDocNormalizer::reopen(BLElement *elemP)
{
    if (closeElem(elemP->getTok()) < 0)
        return -1;

    mDocStreamP->beginElement(elemP);
    mContextVector.back()->push_back(mDocTypeP->getElementDef(elemP->getTok()));
    if (!mDocTypeP->getElementDef(elemP->getTok())->hasHardClose() &&
        mOpenElems.size() > 0)
    {
        return openElements();
    }

    return 0;
}

int
BLDocNormalizer::closeElem(int elemTok)
{
    ElemVector *curContextP = mContextVector.back();
    mOpenElems.clear();

    // if end of context, close it out
    if (mDocTypeP->getElementDef(elemTok)->hasNewContext())
    {
        delete curContextP;
        mContextVector.pop_back();
        return mDocStreamP->endElement(elemTok);
    }

    // close it and elements above it
    for (ElemVector::reverse_iterator ri = curContextP->rbegin();
         ri != curContextP->rend();
         ri++)
    {
        if (mDocStreamP->endElement((*ri)->getTok()) < 0)
            return -1;
        curContextP->pop_back();
        if ((*ri)->getTok() == elemTok)
            break;
        else
            mOpenElems.push_back(mDocTypeP->getElementDef((*ri)->getTok()));
    }

    return 0;
}

int
BLDocNormalizer::openElements()
{
    ElemVector *curContextP = mContextVector.back();

    // if element immediately above it is valid in new element, reopen it, etc
    for (ElemVector::reverse_iterator ri = mOpenElems.rbegin();
         ri != mOpenElems.rend();
         ri++)
    {
        if (curContextP->back()->getShortestPath((*ri)->getTok()) == *ri)
        {
            if (curContextP->back()->hasRequiredAttrs())
                break;
            curContextP->push_back(*ri);
            mElement.setTok((*ri)->getTok());
            if (mDocStreamP->beginElement(&mElement) < 0)
                return -1;
        }
    }

    if (closeElem(elemTok) < 0)
        return -1;
    if (!mDocTypeP->getElementDef(elemTok)->hasHardClose() &&
        mOpenElems.size() > 0 &&
        curContextP->size() > 0)
    {
        // check to see if we can get there from here
        BLElementDef *topElemP = mDocTypeP->getElementDef(curContextP->back()->getTok());
        topElemP = topElemP->getShortestPath(elemP->getTok());
        if (topElemP == NULL)
            return 0;

        // traverse the path
        while (topElemP->getTok() != elemP->getTok())
        {
            if (topElemP->hasRequiredAttrs())
                return 0;
            mElement.setTok(topElemP->getTok());
            mDocStreamP->beginElement(&mElement);
            if (topElemP->hasNewContext())
            {
                curContextP = new ElemVector();
                mContextVector.push_back(curContextP);
            }
            curContextP->push_back(topElemP);
            topElemP = topElemP->getShortestPath(elemP->getTok());
        }

        if (topElemP->hasNewContext())
        {
            curContextP = new ElemVector();
            mContextVector.push_back(curContextP);
        }

        // check to see if it should be empty
        if (elemP->isEmpty() == false)
        {
            if (topElemP->isEmpty() == true)
                elemP->setEmpty(true);
            else
                curContextP->push_back(topElemP);
        }

        return mDocStreamP->beginElement(elemP);
    }

    int
    BLDocNormalizer::endElement(int elemTok)
    {
        if (mContextVector.size() == 0)
            return 0;
        if (mPending && mPendingElement.getTok() == elemTok)
        {
            mPending = false;
            ElemVector *curContextP = mContextVector.back();
            // check that it is open
            bool open = false;
            ElemVector::reverse_iterator ri;
            for (ri = curContextP->rbegin();
                 ri != curContextP->rend();
                 ri++)
            {
                if ((*ri)->getTok() == elemTok)
                {
                    open = true;
                    break;
                }
            }
            if (!open)
                return 0;
        }

        if (closeElem(elemTok) < 0)
            return -1;
        if (!mDocTypeP->getElementDef(elemTok)->hasHardClose() &&
            mOpenElems.size() > 0 &&
            curContextP->size() > 0)
    {

```

```

        return -1;
    }
    else
        break;
}

return 0;
}

// MALConduit.h
#ifdef _MALCONDUIT
#define _MALCONDUIT_

#include "MALWriter.h"
#include "MALIDS.h"
#include "BLClientInfo.h"

class MALConduitManager;

class MALConduit
{
public:
    MALConduit() {}
    virtual ~MALConduit() {}

    virtual void sendInitialResponse(MALWriter *writerP, BLClientInfo *infoP) = 0;
    virtual void sendNormalResponse(MALWriter *writerP, BLClientInfo *infoP) = 0;

    virtual void openDatabase(char *dbName) {}
    virtual void closeDatabase() {}
    virtual void unknownDatabase(char *dbName) {}
    virtual void newIDS(MALIDS *idsP) {}
    virtual void record(int32 uid,
                        AGRecordStatus mod,
                        int32 recordDataLength,
                        void *recordData,
                        int32 platformDataLength,
                        void *platformData) {}

};

class MALConduitFactory
{
public:
    virtual ~MALConduitFactory() {}

    // called after all options have been set
    virtual void initialize(MALConduitManager *) {}
    virtual void finalize() {}

    virtual const char* name() = 0;
    virtual int setOption(const char *name, const char *value) {return 0;}

    virtual MALConduit* makeConduit() = 0;
};

#ifdef _MALCONDUIT_
#define _MALCONDUITMANAGER_
#endif

#include <vector>
#include "ace/Hash_Map_Manager.h"
#include "ace/Synch.h"
#include "MALConduit.h"

typedef ACE_Hash_Map_Manager<const char*, int, ACE_Null_Mutex> MALDatabaseMap;

class MALConduitDispatcher;

// registers and manages conduit factories
class MALConduitManager
{
public:

```

```

static MALConduitManager* instance();

void initialize();
void finalize();

// called by MALConfig.
MALConduitFactory* activateFactory(const char *name); // returns factoryIndex
void registerDatabase(const char *name, MALConduitFactory *factoryP);

MALConduitDispatcher* makeDispatcher();
int getIndexForDatabase(const char *name);

protected:
    MALConduitManager();
    ~MALConduitManager();

    static MALConduitFactory* sFactoryDefs[];
    static MALConduitManager* sInstanceP;

    std::vector<MALConduitFactory*> mActiveFactories;
    MALDatabaseMap mDatabaseTable;
};

// manages conduits for an instance of MALSession
class MALConduitDispatcher
{
public:
    friend class MALConduitManager;
    ~MALConduitDispatcher();

    MALConduit* getConduitForDatabase(const char *name);
    void sendInitialResponses(MALWriter *writerP, BLClientInfo *infoP);
    void sendNormalResponses(MALWriter *writerP, BLClientInfo *infoP);

protected:
    MALConduitDispatcher() {}
    MALConduitDispatcher(std::vector<MALConduit*> *conduitsP,
                        MALConduitManager *managerP);

    std::vector<MALConduit*> *mConduitsP;
    MALConduitManager *mManagerP;
};

#ifdef _MALCONDUITMANAGER_
#define _MALCONDUITMANAGER_
#endif

#ifdef _MALCONFIG_
#define _MALCONFIG_
#endif

class MALConduitManager;
class MALConduitFactory;

class MALConfig
{
public:
    ~MALConfig();

    int readConfigFile(char *fileName);

    int getPort() {return mPort;}
    const char* getInterface() {return mInterface;}

protected:
    static void xmlStartElementHandler( void *userData,
                                        const char *name,
                                        const char **atts);

    static void xmlEndElementHandler(void *userData,
                                    const char *name);

    void unknownOption(const char *option);

    int mPort;
    int mInterface;

```

```

char *mInterface;
MALConduitManager *mConduitManagerP;
MALConduitFactory *mCurrentFactoryP;
};

#ifdef __MALCONFIG__
#define MALCONFIG__ */// MALConstants.h

#define KMALDefaultFilename "mal.config"
#define KMALDefaultPort 80
#define KMALMaxCommands 10000
#define KFileBufSize 4096// MALIDs.h

#ifdef __MALIDS__
#define __MALIDS__
#include "AGProtocol.h"

// wrapper for AGArray with ids in it
class MALIDS
{
public:
    MALIDS(AGArray *arrayP)
    : mArrayP(arrayP), mIndex(0) {}

    // reset the iterator
    void rewind()
    {
        mIndex = 0;
    }

    // returns number of id pairs
    int count()
    {
        return mArrayP->count / 2;
    }

    // returns true if id's were updated / false if already at end of array
    bool next(int32 *oldidP, int32 *newidP)
    {
        if (mIndex < mArrayP->count)
        {
            *oldidP = (int32)AGArrayElementAt(mArrayP, mIndex);
            *newidP = (int32)AGArrayElementAt(mArrayP, mIndex+1);

            mIndex += 2;
            return true;
        }
        return false;
    }

protected:
    AGArray *mArrayP;
    int mIndex;
};

#endif // __MALIDS__ */// MALIO.h

#ifdef __MALIO__
#define __MALIO__
#include "ace/OS.h"

#ifdef defined (ACE_LACKS_PRAGMA_ONCE)
#pragma once
#endif // ACE_LACKS_PRAGMA_ONCE *//

#include "ace/sock_Stream.h"
#include "ace/Iostream.h"
#include "AGReader.h"
#include "AGWriter.h"
#include "MALTypes.h"

class MALIO
{
public:
    static void init(ACE_SOCK_Stream *streamP, AGReader *readerP, AGWriter *writerP);

protected:
    static int read(void *data, void *dst, int len);
    static int write(void *data, void *buf, int len);
};

#endif // __MALIO__ */// MALSession.h

#ifdef __MALSESSION__
#define __MALSESSION__
#include "MALSession.h"
#include "AGReader.h"
#include "AGWriter.h"
#include "BLCClientInfo.h"

class MALConduitManager;
class MALConduitDispatcher;
class MALConduit;

class MALSession
{
public:
    MALSession();
    ~MALSession();

    int doSession(ACE_SOCK_Stream *streamP);

protected:
    int readCommands();
    int readHeader();
    int sendResponse();

    int readExpansionCmd(int len);
    int readHelloCmd();
    int readDeviceInfoCmd();
    int readOpenDatabaseCmd();
    int readCloseDatabaseCmd();
    int readRecordCmd();
    int readUnknownDatabaseCmd();
    int readNewIdsCmd();
    int readPingCmd();
    int readXMLDataCmd();

    int8 mMajorVersion;
    int8 mMinorVersion;
    bool mInitialSession;

    AGReader mReader;
    AGWriter mWriter;

    BLCClientInfo mClientInfo;
    MALConduitDispatcher *mDispatcherP;
    MALConduit *mConduitP;
};

#endif // __MALSESSION__ */// MALTypes.h

#ifdef __MALTYPES__
#define __MALTYPES__
#include "ace/INET_Addr.h"
#include "ace/sock_Stream.h"
#include "ace/Iostream.h"

typedef ACE_Iostream<ACE_SOCK_Stream> MAL_Iostream;
#define MAL_Iostream MAL_Iostream, ACE_INET_Addr

#endif // __MALTYPES__ */// MALWriter.h

#ifdef __MALWRITER__
#define __MALWRITER__

```

```

#define __MALWRITER__
#include "AGProtocol.h"
#include "AGPalmProtocol.h"
#include "ace/OS.h"

#define ADD_STRING_LEN(str, strlen) (if (str != NULL) {strlen = ACE_OS::strlen(str); len +=
AGCompactSize(strlen); len += strlen;})

// wrapper for AGProtocol commands conduits should have access to
class MALWriter
{
public:
    friend class MALSession;

    void writeDatabaseConfig(char *dbname,
        AGDBConfigType config,
        AGBool sendRecordPlatformData,
        int32 platformDataLength,
        void *platformData)
    {
        AGWriteDATABASECONFIG(mWriterP,
            dbname,
            config,
            sendRecordPlatformData,
            platformDataLength,
            platformData);
    }

    void writePalmDatabaseConfig(char *dbname,
        AGDBConfigType config,
        AGBool sendRecordPlatformData,
        uint32 creator,
        uint32 type,
        uint32 flags)
    {
        int platformDataLength = AGCompactSize(creator);
        platformDataLength += AGCompactSize(type);
        platformDataLength += AGCompactSize(flags);

        int32 len = 0, dbnameLen = 0;

        ADD_STRING_LEN(dbname, dbnameLen);
        len += AGCompactSize(config);
        len += 1; // 1 boolean - sendRecordPlatformData
        len += AGCompactSize(platformDataLength);
        len += platformDataLength;

        // write the command
        AGWriteCompactInt(mWriterP, AG_DATABASECONFIG_CMD);
        AGWriteCompactInt(mWriterP, len);

        AGWriteString(mWriterP, dbname, dbnameLen);
        AGWriteCompactInt(mWriterP, config);
        AGWriteBoolean(mWriterP, sendRecordPlatformData);
        AGWriteCompactInt(mWriterP, platformDataLength);
        AGPalmWriteDBConfigPlatformData(mWriterP, creator, type, flags);
    }

    void writeOpenDatabase(char *dbname)
    {
        AGWriteOPENDATABASE(mWriterP,
            dbname);
    }

    void writeCloseDatabase()
    {
        AGWriteCLOSEDATABASE(mWriterP);
    }

    void writeDeleteDatabase(char *dbname)
    {
        AGWriteDELETEDATABASE(mWriterP,
            dbname);
    }
}

```

```

void writeClearMods()
{
    AGWriteCLEARMODS(mWriterP);
}

void writeRecord(int32 uid,
    AGRecordStatus mod,
    int32 recordDataLength,
    void *recordData,
    int32 platformDataLength,
    void *platformData)
{
    AGWriteRECORD(mWriterP,
        uid,
        mod,
        recordDataLength,
        recordData,
        platformDataLength,
        platformData);
}

void writeTask(char *currentTask)
{
    AGWriteTASK(mWriterP,
        currentTask,
        false);
}

void writeItem(int32 currentItemNumber,
    int32 totalItemCount,
    char *currentItem)
{
    AGWriteITEM(mWriterP,
        currentItemNumber,
        totalItemCount,
        currentItem);
}

void writeXMLData(int32 dataLen,
    void *dataBytes)
{
    AGWriteXMLDATA(mWriterP,
        dataLen,
        dataBytes);
}

void writeExpansion(int32 expansionCommand,
    int32 commandLength,
    void *commandBytes)
{
    AGWriteEXPANSION(mWriterP,
        expansionCommand,
        commandLength,
        commandBytes);
}

protected:
    MALWriter(AGWriter *writerP) : mWriterP(writerP) {}
    MALWriter() {}
    ~MALWriter() {}

    void setWriter(AGWriter *writerP)
    {
        mWriterP = writerP;
    }

    AGWriter *mWriterP;
};

#endif /* __MALWRITER__ */
// BLClientInfo.h

```

```

char *mCharset;
char *mUserAgent;
char *mAcceptString;
int mAvailableBytes;
int mScreenWidth;
int mScreenHeight;
int mBitDepth;
std::vector<BLTransformManager::ContentType> mAcceptTypes;
};

#endif /* __BLCLIENTINFO__ */ /// BLClientInfoManager.h

#ifndef __BLCLIENTINFOMANAGER__
#define __BLCLIENTINFOMANAGER__

#include "BLClientInfo.h"

class BLClientInfoManager
{
public:
    static BLClientInfoManager* instance();

    void initialize();
    void finalize();

    BLClientInfo* getUserAgentInfo(const char *uaString);

protected:
    BLClientInfoManager();
    ~BLClientInfoManager();

    static BLClientInfoManager* sInstanceP;

    typedef std::pair<char*, BLClientInfo> InfoPair;
    std::vector<InfoPair> mInfoVector;
    BLClientInfo* mDefaultInfoP;
};

#endif /* __BLCLIENTINFOMANAGER__ */ /// BLLog.h

#ifndef __BLLOG__
#define __BLLOG__

#include "ace/OS.h"
#include "ace/synch.h"
#include "ace/sock_Stream.h"
#include <fstream>

class BLLog
{
protected:
    BLLog(const char *path);

    void writeTimestamp()
    {
        time_t now = ACE_OS::time();
        char *timeStr = ACE_OS::ctime(&now);
        mFile << " ";
        mFile.write(timeStr, ACE_OS::strlen(timeStr) - 1);
        mFile << " ";
    }

    const char *mPath;
    std::ofstream mFile;
    ACE_Thread_Mutex mMutex;
};

class BLAccessLog : public BLLog
{
public:
    static int initialize();
    static void finalize();

    static void write(char* userAgent, ACE_SOCK_Stream *streamP, char *path)

```

```

char *mUsername();
int getAvailableBytes();
int getBitDepth();
char* getOSName();
char* getOSVersion();
char* getSerialNumber();
char* getLanguage();
char* getCharset();
int getScreenWidth();
int getScreenHeight();
char* getUserAgent();
char* getAcceptString();
int getUsableScreenWidth();
{
    // depending on user-agent, modify screen width
    // no checks
    return mScreenWidth - 10;
}

int getUsableScreenHeight()
{
    // depending on user-agent, modify screen height
    return mScreenHeight - 10;
}

std::vector<BLTransformManager::ContentType>* getAcceptTypes() {return
    mAcceptTypes;}

    void setUsername(char *username)
    {mUsername = username;}
    void setAvailableBytes(int availableBytes) {mAvailableBytes = availableBytes;}
    void setBitDepth(int bitDepth)
    {mBitDepth = bitDepth;}
    void setOSName(char *osName)
    {mOSName = osName;}
    void setOSVersion(char *osVersion)
    {mOSVersion = osVersion;}
    void setSerialNumber(char *serialNumber)
    {mSerialNumber = serialNumber;}
    void setLanguage(char *language)
    {mLanguage = language;}
    void setCharset(char *charset)
    {mCharset = charset;}
    void setScreenWidth(int width)
    {mScreenWidth = width;}
    void setScreenHeight(int height)
    {mScreenHeight = height;}
    void setUserAgent(char *userAgent)
    {mUserAgent = userAgent;}
    void setAcceptString(char *acceptString)
    {mAcceptString = acceptString;}
    void setAcceptType(BLTransformManager::ContentType type)
    {mAcceptTypes.push_back(type);}

protected:
    char *mUsername;
    char *mOSName;
    char *mOSVersion;
    char *mSerialNumber;
    char *mLanguage;

```



```

{
    sinstanceP->writeRecord(userAgent, streamP, path);
}

protected:
    BLAccessLog();
    void writeRecord(char* userAgent, ACE_SOCK_Stream *streamP, char *path)
    {
        if (userAgent == NULL)
            userAgent = "(none)";

        ACE_INET_Addr addr;
        streamP->get_remote_addr(addr);

        mMutex.acquire();

        writeTimestamp();
        mFile << addr.get_host_addr() << " " << userAgent << " " << path << endl;

        mMutex.release();
    }

    static BLAccessLog *sinstanceP;

class BLErrorLog : public BLLog
{
public:
    static int initialize();
    static void finalize();

    static void write(char* message)
    {
        sinstanceP->writeRecord(message);
    }
};

protected:
    BLErrorLog();
    void writeRecord(char* message)
    {
        mMutex.acquire();

        writeTimestamp();
        mFile << message << endl;

        mMutex.release();
    }

    static BLErrorLog *sinstanceP;

};

#endif /* BLLOG */
// BLUtils.h

#ifndef BLUTILS_H
#define BLUTILS_H

class BLUtils
{
public:
    static int min(int x, int y)
    {
        return x < y ? x : y;
    }

    static char caseless(char c)
    {
        return c & 0xDF;
    }

    static char* strdup(const char* str)
    {
        char *newStr = new char[ACE_OS::strlen(str) + 1];
        if (newStr != NULL)

```

```

    ACE_OS::strcpy(newStr, str);
};

#endif /* BLUTILS_H */
// BLHttpAcceptor.h

#ifndef BLHTTPACCEPTOR_H
#define BLHTTPACCEPTOR_H

#include "ace/Acceptor.h"
#include "ace/SOCK_Acceptor.h"
#include "BLHttpHandler.h"

class ACE_Svc_Export BLHttpAcceptor : public ACE_Strategy_Acceptor<BLHttpHandler,
    ACE_SOCK_ACCEPTOR>
{
protected:
    // = Service configurator hooks.
    virtual int init(int argc, char *argv[]);
    // Perform initialization.

    virtual int fini();
    // cleanup for shutdown

    virtual int make_svc_handler (BLHttpHandler *h);
    // make a new handler

    int parseArgs (int argc, char *argv[]);
    // parse command line args

    ACE_Schedule_All_Threaded_Strategy<BLHttpHandler> mScheduleStrategy;

    char *mConfigFilename;
};

ACE_SVC_FACTORY_DECLARE (BLHttpAcceptor)

#endif /* BLHTTPACCEPTOR */
// BLHttpBase.h

#ifndef BLHTTPBASE_H
#define BLHTTPBASE_H

#include "ace/SOCK_Stream.h"
#include "BLHttpRequestInfo.h"

class BLTransportStream;

class BLHttpBase
{
public:
    BLHttpBase();

    void init(ACE_SOCK_Stream *streamP, ACE_Time_Value *timeoutP);
    int readMessage(BLTransportStream *transportP);

    int getMajorVersion() (return mMajorVersion);
    int getMinorVersion() (return mMinorVersion);

protected:
    virtual int handleHeader(const char* name, const char* value) {return 0;}

    int parseHeaders();
    int sendString(const char* str);
    int sendHeader(const char* header, const char* value);
    int sendHeader(const char* header, int);
    int sendHeaders(BLHttpRequestInfo *info);
    int readLine(char *buf, int size);
    int readChunk(int length, BLTransportStream *transportP);
    int readChunkedEncoding(BLTransportStream *transportP);
    int readFull(BLTransportStream *transportP);

    ACE_SOCK_Stream *mStreamP;

```

```

ACE_Time_Value mTimeout;

BLTransformManager::ContentType mContentType;
int mContentLength;
int mMajorVersion;
int mMinorVersion;
bool mChunkedEncoding;
};

#endif /* __BLHTTPBASE__ */ /// BLHttpHandler.h

#include "ace/Synch.h"
#include "ace/Svc_Handler.h"
#include "ace/Socket_Stream.h"
#include "BLHttpBase.h"
#include "BLTransportStream.h"
#include "BLHttpRequestInfo.h"

#ifdef ACE_WIN32
#define HTTP_IOSTREAM ACE_SOCKET_Stream, ACE_INET_Addr
#else
#define HTTP_IOSTREAM ACE_SOCKET_Stream //, ACE_INET_Addr
#endif //ACE_WIN32

class BLHttpHandler : public ACE_Svc_Handler<HTTP_IOSTREAM, ACE_NULL_SYNCH>, public
BLHttpBase, public BLTransportFeeder
{
    // = TITLE
    // = Product object created by an <MALAcceptor>. An
    // = DESCRIPTION
    // = Instantiates and processes MALSessions
public:
    typedef enum {
        kGettype,
        kPosttype,
    } RequestType;

    BLHttpHandler(ACE_Thread_Manager *threadManagerP = 0);
    ~BLHttpHandler();

    virtual int open (void * = NULL);
    virtual int svc (void);

    RequestType gettype()
    { return mType; }
    const char* getUserAgent()
    { return mUserAgent; }
    BLHttpRequestInfo* getRequestInfo()
    { return mRequestInfo; }

protected:
    virtual int feed(BLTransportStream *transportP);
    virtual int handleHeader(const char* name, const char* value);

    int readRequestLine(char *pathBuffer,
                        int bufferSize);

    RequestType mType;
    BLHttpRequestInfo mRequestInfo;
    char *mUserAgent;
};

#endif /* __BLHTTPHANDLER__ */
// BLHttpRequest.h

#include "HTTPREQUEST__"
#define __HTTPREQUEST__

#include "ace/Socket_Stream.h"
#include "ace/Socket_Connector.h"
#include "BLTransportStream.h"
#include "BLTransportRequest.h"
#include "BLHttpBase.h"
#include "BLUtils.h"

```

```

class BLHttpRequest : public BLTransportRequest, protected BLHttpBase, protected
BLTransportStream
{
public:
    BLHttpRequest();
    virtual ~BLHttpRequest();

    void init(ACE_SOCKET_Stream *streamP);
    int requestFile(BLUrl *urlP, BLClientInfo *clientInfoP, BLHttpRequestInfo *infoP);
    virtual int readFile(BLTransportStream *transportP);

    virtual int getContentLength();

    virtual BLTransformManager::ContentType getContentType()
    { return mContentType; }

    virtual BLHttpRequestInfo* getRequestInfo()
    { return mRequestInfo; }
    char* getLocation()
    { return mLocation; }

protected:
    virtual int init(int length, BLTransformManager::ContentType type);
    virtual int put(char *data, int size, bool final);
    virtual int handleHeader(const char* name, const char* value);

    int readStatusLine();

    BLHttpRequestInfo *mProxyInfoP;
    BLHttpRequestInfo mRequestInfo;
    char *mLocation;
};

class BLHttpRequestFactory : public BLTransportRequestFactory
{
public:
    virtual void initialize(BLTransportManager *);
    virtual void finalize();

    virtual const char* getName();
    virtual BLTransportRequest* requestFile(BLUrl *urlP, BLClientInfo *clientInfoP,
    BLHttpRequestInfo *infoP);

protected:
    ACE_SOCKET_Connector mConnector;
};

#endif /* __HTTPREQUEST__ */ /// BLHttpRequestInfo.h

#include "BLHTTPREQUESTINFO__"
#define __BLHTTPREQUESTINFO__

#include <vector>
#include <utility>
#include "ace/OS.h"
#include "BLTransportStream.h"
#include "BLUtils.h"

class BLHttpRequestInfo
{
public:
    typedef std::pair<char*, char*> HeaderPair;
    typedef std::vector<HeaderPair> HeaderVector;

    BLHttpRequestInfo()
    { mPostP(NULL); }

    ~BLHttpRequestInfo()
    { clear(); }
};

```

```

void clear()
{
    mPostP = NULL;
    for (HeaderVector::iterator i = mHeaders.begin();
        i != mHeaders.end();
        i++)
    {
        delete i->first;
        delete i->second;
    }
    mHeaders.clear();
}

void setPostData(BLTransportFeeder *postP)
{
    mPostP = postP;
}

BLTransportFeeder* getPostData()
{
    BLTransportFeeder* dataP = mPostP;
    mPostP = NULL;
    return dataP;
}

void setHeader(const char *name, const char* value)
{
    HeaderPair pair;
    pair.first = BUUtils::strup(name);
    pair.second = BUUtils::strup(value);
    mHeaders.push_back(pair);
}

HeaderVector* getHeaders()
{
    return &mHeaders;
}

protected:
    BLTransportFeeder *mPostP;
    HeaderVector mHeaders;
};

#endif /* __BLHTTPREQUESTINFO__ */ /// BLProxySession.h

#ifdef __BLPROXYSESSION__
#define __BLPROXYSESSION__
#include "BLHttpHandler.h"
#include "BLUrl.h"
class BLClientInfo;
class ProxyStream;
class BLProxySession
{
public:
    BLProxySession();

    int doSession(char *path, BLHttpHandler *handlerP, ACE_SOCK_Stream *ios);

protected:
    int parsePath(char *path, BLUrl *urlP);
    int error(char *error, BLHttpHandler *handlerP, BLClientInfo *clientInfoP,
        ACE_SOCK_Stream *ios);
    ACE_SOCK_Stream* getProxyStream(BLHttpHandler *handlerP, ACE_SOCK_Stream *ios,
        BLHttpRequestInfo *infoP);
};

#endif /* __BLPROXYSESSION__ */ /// BLTransportManager.h

#ifdef __BLTRANSPORTMANAGER__

```

```

#define __BLTRANSPORTMANAGER__
#include "BLUrl.h"
#include "BLTransportRequest.h"

class BLClientInfo;

class BLTransportManager
{
public:
    static BLTransportManager* instance();

    void initialize();
    void finalize();

    // returns NULL on error
    BLTransportRequest* requestFile(BLUrl *urlP, BLClientInfo *clientInfoP,
        BLHttpRequestInfo *infoP);

    void registerProtocolHandler(BLUrl::ProtocolType protocol, BLTransportRequestFactory
        *factoryP);

protected:
    BLTransportManager();
    ~BLTransportManager();

    BLTransportRequestFactory* getFactoryForProtocol(BLUrl::ProtocolType protocol);

    static BLTransportManager* s_instanceP;
    static BLTransportRequestFactory* s_factoryDefs[];

    BLTransportRequestFactory *mHandlerTable[BLUrl::KNumProtocols];
};

#endif /* __BLTRANSPORTMANAGER__ */ /// BLTransportRequest.h

#ifdef __BLTRANSPORTREQUEST__
#define __BLTRANSPORTREQUEST__
#include "BLUrl.h"
#include "BLTransportStream.h"
#include "BLHttpRequestInfo.h"

class BLTransportManager;
class BLTransportStream;
class BLClientInfo;

class BLTransportRequest
{
public:
    virtual ~BLTransportRequest() {}

    virtual int readFile(BLTransportStream *transportP) = 0;

    // return NULL if unknown
    virtual BLTransformManager::ContentType getContentType() = 0;
    // return -1 if unknown
    virtual int getContentLength() = 0;

    virtual BLHttpRequestInfo* getRequestInfo() { return NULL; }
};

class BLTransportRequestFactory
{
public:
    virtual ~BLTransportRequestFactory() {}

    // called after all options have been set
    virtual void initialize(BLTransportManager *) {}
    virtual void finalize() {}

    virtual const char* getname() = 0;
    virtual int setOption(const char *name, const char *value) { return 0; }
};

```

```

virtual BLTransportRequest* requestFile(BLURL *urlp, BLClientInfo *clientInfo,
BLHttpRequestInfo *info) = 0;

#endif /* __BLTRANSPORTREQUEST__ */ /// BLTransportStream.h

#ifndef __BLTRANSPORTSTREAM__
#define __BLTRANSPORTSTREAM__

#include "BLTransformManager.h"

class BLTransportStream
{
public:
    enum {UnknownLength = -1};

    virtual int init(int length, BLTransformManager::ContentType type) = 0;
    virtual int put(char *data, int size, bool final) = 0;
};

class BLTransportFeeder
{
public:
    virtual int feed(BLTransportStream *transportP) = 0;
};

class BLAccumulatingStream : public BLTransportStream
{
public:
    BLAccumulatingStream();
    virtual ~BLAccumulatingStream();

    virtual int init(int length, BLTransformManager::ContentType type);
    virtual int put(char *data, int size, bool final);
    virtual int putfinal(char *data, int size) = 0;

protected:
    char *mDataBuf;
    int mCurrentDataSize;
    int mTotalDataSize;
    BLTransformManager::ContentType mType;
};

// a black hole
class BLNullTransportStream : public BLTransportStream
{
public:
    virtual int init(int length, BLTransformManager::ContentType type) {return 0;}
    virtual int put(char *data, int size, bool final) {return 0;}
};

#endif /* __BLTRANSPORTSTREAM__ */ /// BLTransportWriter.h

#ifndef __BLTRANSPORTWRITER__
#define __BLTRANSPORTWRITER__

#include <ostream.h>

class BLTransportStream;
#define kStreamBufSize 4096
class BLTransportStreamBuf : public ostreambuf
{
public:
    BLTransportStreamBuf(BLTransportStream *streamP);
    virtual ~BLTransportStreamBuf();

    void setTransportStream(BLTransportStream *streamP) {mStreamP = streamP;}

protected:
    virtual int overflow(int c = EOF);
    virtual int sync();
    virtual int underflow();
};

char mBuf[kStreamBufSize];
BLTransportStream *mStreamP;

class BLTransportWriter : public ostream
{
public:
    BLTransportWriter()
        : mStreamBuf(NULL),
          ostream(&mStreamBuf)
    {
    }

    BLTransportWriter(BLTransportStream *streamP)
        : mStreamBuf(streamP),
          ostream(&mStreamBuf)
    {
    }

    void setTransportStream(BLTransportStream *streamP)
    {mStreamBuf.setTransportStream(streamP);}

protected:
    BLTransportStreamBuf mStreamBuf;
};

#endif /* __BLTRANSPORTWRITER__ */ /// BLURL.h

#ifndef __BLURL__
#define __BLURL__

#include "ace/inet_addr.h"

class BLURL
{
public:
    BLURL();
    ~BLURL();

    typedef enum {
        UnknownProtocol = -1,
        HttpProtocol,
        KNumProtocols
    } ProtocolType;

    int set(ProtocolType protocol,
            char *hostName,
            char *path,
            int port);

    int set(ProtocolType protocol,
            char *hostName,
            char *path);

    int set(char *urlString);
    int set(ProtocolType protocol, char *urlString);

    int setRelative(BLURL *baseURLP, char *path);
    void setPort(int port);

    const char* getHostName() {return mHostName;}
    const char* getPath() {return mPath;}
    ProtocolType getProtocol() {return mProtocol;}
    ACE_INET_Addr* getAddr() {return &mAddr;}

    char* toString();

    static bool isRelative(char *urlString);
    static bool isAbsolute(char *urlString) { return !isRelative(urlString); }

    static ProtocolType stringToProtocolType(char *protocolString);

protected:
    struct ProtocolInfo

```

```

{
    char *string;
    int defaultPort;

};

void clear();
static ProtocolInfo sProtocolTable[kNumProtocols];

ProtocolType mProtocol;
char *mHostName;
char *mPath;
char *mString;
ACE_INET_Addr mAddr;
};

#endif /* _BLURL_ */
#endif BLIMAGETRANSFORMER_H
#endif BLIMAGETRANSFORMER_H

#include <utility>
#include <vector>

#include "BLTransformation.h"
#include "BLTransportStream.h"

/* Define FilterType HELLO
#define PROT_READ HELLO1
#define PROT_WRITE HELLO2
#define MAP_SHARED HELLO3
#define MAP_PRIVATE HELLO4
#define DIR_HELLOS */
#endif PROT_READ
#endif PROT_WRITE
#endif MAP_SHARED
#endif MAP_PRIVATE
#endif DIR

#define FilterType HELLO
#define DIR_HELLO2
#define dirent HELLO3
#include <magick/magick.h>
#include <magick/image.h>

#if defined(_cplusplus) || defined(c_plusplus)
#endif class
#endif

class BLImageTransformer : public BLAccumulatingStream, public BLTransformation
{
public:
    BLImageTransformer(BLTransformationManager::ContentType fromType,
        BLTransformationManager::ContentType toType);

    virtual ~BLImageTransformer();

    int init(BLTransportStream *outputTransportP,
        BLImageTransformerFactory *parentFactoryP);

    virtual BLTransportStream* getTransportStream();

    virtual int putFinal(char *data, int size);

    void setClientInfo(BLClientInfo *cip);

private:
    int readMem (const char *blobP, const size_t length);
    int writeImage();
    int transformImage();
    int writeFalmBitmap();

```

```

BLTransportStream *mOutputTransportP;
BLImageTransformerFactory *mParentFactoryP;
Image *mImage;
ImageInfo mImageInfo;

BLClientInfo *mClientInfo;

int mOutputPath;
int mOutputV;
int mOutputIsColor;
int mOutputQuality;
};

class BLImageTransformerFactory : public BLTransformationFactory
{
public:
    BLImageTransformerFactory() : BLTransformationFactory() {}
    ~BLImageTransformerFactory();

    virtual int initialize(BLTransformationManager *);
    virtual const char* getName();

    virtual BLTransformation* makeTransformation(BLTransformationManager::ContentType fromType,
        BLTransformationManager::ContentType toType,
        BLClientInfo *infoP,
        BLTransportStream *streamP);
};

#endif
// BLDocNormalizer.h

#ifdef _BLDOCNORMALIZER_H_
#define _BLDOCNORMALIZER_H_

#include "BLDocStream.h"
#include "BLDocType.h"
#include "ace/config.h"
#include <vector>

#ifdef ACE_WIN32
#pragma warning(disable:4786)
#endif

class BLDocNormalizer : public BLDocStream
{
public:
    BLDocNormalizer(BLDocStream *streamP);

    virtual int beginDoc(BLDocType *docTypeP);
    virtual int endDoc();

    virtual int beginElement(BLElement *elemP);
    virtual int endElement(int elemTok);
    virtual int charData(const char* data, int length);

protected:
    int beginElement_i(BLElement *elemP);
    int openElement(BLElement *elemP);
    int reopen(BLElement *elemP);
    int closeElement(int elemTok);
    int openElements();

    typedef std::vector<BLElement> ElemVector;

    struct Context
    {
        ElemVector mOpenElements;
        ElemVector mClosedElements;
    };

    typedef std::vector<Context*> ContextVector;
    ContextVector mContextVector;

```

```

    ElemVector mPendingElements;
    BLElement mElement;

    BLDocStream *mDocStreamP;
    BLDocType *mDocTypeP;

    const char* mCData;
    int mCDataLength;

}

#endif /* _BLDOCNORMALIZER_H_ */ // BLDocStream.h

#ifdef _BLDOCSTREAM_H_
#define _BLDOCSTREAM_H_

#include <vector>

class BLDocType;
class BLElement;
class BLAttribute;

class BLDocStream
{
public:
    virtual ~BLDocStream() {}
    virtual int beginDoc(BLDocType *docTypeP) = 0;
    virtual int endDoc() = 0;

    virtual int beginElement(BLElement *elemP) = 0;
    virtual int endElement(int elemTok) = 0;
    virtual int charData(const char* data, int length) = 0;
};

class BLAttribute
{
public:
    enum Type
    {
        kTokenType,
        kIntType,
        kCDataType,
        kBoolType,
        kLengthType,
        kNoValueType
    };

    BLAttribute();
    ~BLAttribute();
    BLAttribute& operator= (const BLAttribute &attr);

    int cloneString(); // make local copy of string

    void setTok(int attrTok)
    {
        mTok = attrTok;
    }

    int getTok()
    {
        return mTok;
    }

    Type getType()
    {
        return mType;
    }

    void setType(Type type)
    {
        mType = type;
    }

    void setTokValue(int tokValue);
    void setIntValue(int intValue);
    void setCDataValue(char* cdataValue);
    void setBoolValue(bool boolValue);

    int getTokValue()
    {
        return mValue.tokValue;
    }

    int getIntValue()
    {
        return mValue.intValue;
    }

    char* getCDataValue()
    {
        return mValue.cdataValue;
    }

    bool getBoolValue()
    {
        return mValue.boolValue;
    }

protected:
    union Value
    {
        int tokValue;
        int intValue;
        char* cdataValue;
    }
};

bool boolValue;

type mType;
int mTok;
bool mOwnString;
Value mValue;

};

class BLElement
{
public:
    typedef std::vector<BLAttribute> AttrVector;

    BLElement();
    BLElement& operator= (BLElement &elem);

    void clear();

    void setTok(int elemTok)
    {
        mTok = elemTok;
    }

    int getTok()
    {
        return mTok;
    }

    void setEmpty(bool empty)
    {
        mEmpty = empty;
    }

    bool isEmpty()
    {
        return mEmpty;
    }

    void addAttribute(BLAttribute *attrP);
    BLAttribute* getAttribute(int attrTok);
    AttrVector* getAttrVector();

protected:
    int mTok;
    bool mEmpty;
    AttrVector mAttrVector;
};

#include "BLDocStream_i.h"

#endif /* _BLDOCSTREAM_H_ */ // BLDocStream_i.h
// DO NOT INCLUDE THIS FILE DIRECTLY

#include "ace/OS.h"
#include "BLUtils.h"

inline
BLAttribute::BLAttribute()
: mOwnString(false)
{
}

inline
BLAttribute::~BLAttribute()
{
    if (mOwnString)
        delete[] mValue.cdataValue;
}

inline BLAttribute&
BLAttribute::operator= (const BLAttribute &attr)
{
    mOwnString = false;
    mType = attr.mType;
    mTok = attr.mTok;
    mValue.cdataValue = attr.mValue.cdataValue;
    return *this;
}

inline int
BLAttribute::cloneString()
{
    if (mType == kCDataType)
    {
        ACE_ALLOCATOR_RETURN(mValue.cdataValue,
            BUJUtils::strdup(mValue.cdataValue),
            -1);
    }
}

```

```

        mOwnString = true;
    }
    return 0;
}

inline void
BLAttribute::setTokValue(int tokValue)
{
    if (mOwnString)
    {
        delete[] mValue.cdataValue;
        mOwnString = false;
    }
    mType = kTokenType;
    mValue.tokValue = tokValue;
}

inline void
BLAttribute::setIntValue(int intValue)
{
    if (mOwnString)
    {
        delete[] mValue.cdataValue;
        mOwnString = false;
    }
    mType = kIntType;
    mValue.intValue = intValue;
}

inline void
BLAttribute::setCDATAValue(char* cdataValue)
{
    if (mOwnString)
    {
        delete[] mValue.cdataValue;
        mOwnString = false;
    }
    mType = kCDATAtype;
    mValue.cdataValue = cdataValue;
}

inline void
BLAttribute::setBoolValue(bool boolValue)
{
    if (mOwnString)
    {
        delete[] mValue.cdataValue;
        mOwnString = false;
    }
    mType = kBoolType;
    mValue.boolValue = boolValue;
}

inline
BLElement::BLElement()
: mEmpty(false)
{
}

inline void
BLElement::clear()
{
    mEmpty = false;
    mAttrVector.clear();
}

inline BLElement&
BLElement::operator= (BLElement &elem)
{
    mEmpty = elem.mEmpty;
    mTok = elem.mTok;
    mAttrVector = elem.mAttrVector;

    for (AttrVector::iterator i = mAttrVector.begin();

```

```

        i := mAttrVector.end();
        i++;
    }
    return *this;
}

inline void
BLElement::addAttribute(BLAttribute *attrP)
{
    mAttrVector.push_back(*attrP);
}

inline BLAttribute*
BLElement::getAttribute(int attrTok)
{
    for (AttrVector::iterator i = mAttrVector.begin();
        i != mAttrVector.end();
        i++)
    {
        if (i->getTok() == attrTok)
            return &(*i);
    }
    return NULL;
}

// BLDocTransformer.h
#ifdef _BLDOCTRANSFORMER_H_
#define _BLDOCTRANSFORMER_H_
#include "BLTransformation.h"
#include "BLHtmlDocType.h"
#include "BLMlDocType.h"
#include "BLMmlcTable.h"
class BLDocParser;
class BLDocStream;
class BLDocTransformer : public BLTransformation
{
public:
    friend class BLDocTransformerFactory;

    BLDocTransformer(BLTransformation::ContentType fromType,
        BLTransformation::ContentType toType);
    virtual ~BLDocTransformer();

    int init(BLTransportStream *streamP);
    virtual BLTransportStream* getTransportStream();

protected:
    BLTransportStream *mParserP;
    BLDocStream *mNormalizerP;
    BLDocStream *mAdapterP;
    BLDocStream *mOutputP;

    static BLHtmlDocType* sHtmlDocType;
    static BLMlDocType* sMlDocType;
    static BLMmlcTable* sMmlcTable;
};

class BLDocTransformerFactory : public BLTransformationFactory
{
public:
    virtual int initialize(BLTransformation *);
    virtual const char* getName();

    virtual BLTransformation* makeTransformation(BLTransformation::ContentType fromType,
        BLTransformation::ContentType toType,
        BLClientInfo *infoP,
        BLTransportStream *streamP);

```

```

protected:
    BLHtmlDocType mHtmlDocType;
    BLMmlDocType mMmlDocType;
    BLMmlTable mMmlTable;
};

#ifdef __BLDOCTransformer_H__
// BLDocType.h
#define __BLDOCTYPE_H__
#include "BLTokenTable.h"
#include "BLDocStream.h"
#include <vector>

class BLAttributeDef;
class BLElementDef;

class BLDocType
{
public:
    enum {kDataElement = 0};

    BLDocType(int numElems);
    virtual ~BLDocType();

    virtual int initialize() = 0;

    BLElementDef* getElementDef(int elemTok);
    bool validateElement(BLElement *elem);

    int getElementTok(char* str);
    const char* getElementString(int elemTok);

    int getAttributeTok(char* str);
    const char* getAttributeString(int attrTok);

    int getValueTok(char* str);
    const char* getValueString(int valueTok);

    int getEntityTok(char* str);

    virtual const char* getDocTypeString() = 0;
    virtual BLElementDef* getRootElementDef() = 0;

protected:
    int initTables(int numElems, int numAttrs, int numValues);
    int initPaths();

    void addClassToElem(int elem, int elemClass);
    void addClassToClass(int superClass, int subClass);
    void addElemsToClass(int toClass, int elemClass);

    void addElementTok(char* string, int tok);
    void addAttributeTok(char* string, int tok);
    void addValueTok(char* string, int tok);
    void addEntityTok(char* string, int tok);

    BLAttributeDef* makeAttrDef(int attrTok, BLAttribute::Type attrType, bool required);

    int mNumElements;
    char **mElementStrings;
    BLElementDef *mElementDefs;
    BLTokenTable mElementToks;

    char **mAttributeStrings;
    BLTokenTable mAttributeToks;
    std::vector<BLAttributeDef*> mAttributeDefs;

    char **mValueStrings;
    BLTokenTable mValueToks;

    BLTokenTable mEntityToks;
};

```

```

class BLElementDef
{
friend class BLDocType;

public:
    int getTok();
    bool hasRequiredAttrs();
    bool hasNewContext();
    bool hasHardClose();
    bool isEmpty();
    bool isEmptyOK();
    int getNumAttrs();
    bool getIgnoreWhitespace();
    bool getResetWhitespace();
    bool getPreserveWhitespace();
    BLElementDef* getShortestPath(int elemTok) {return mElemPathP[elemTok];}

    bool validateElement(BLElement *elem);

    void setNewContext();
    void setHardClose();
    void setPreserveWhitespace();
    void setIgnoreWhitespace();
    void setResetWhitespace();
    void setEmptyOK(bool ok);
    void addToClass(int elemClass);
    int getClass();
    void addAttribute(BLAttributeDef *attrP);
    void addElement(BLElementDef *elemP);
    void removeElement(BLElementDef *elemP);

protected:
    BLElementDef();
    ~BLElementDef();
    int initShortestPath(BLElementDef* elemP);
    int init(int elemTok, int numElems, int numAttrs);

    int mClass;
    int mNumAttrs;
    int mTok;
    bool mNewContext;
    bool mHardClose;
    bool mInPath;
    bool mEmptyOK;
    bool mPreserveWhitespace;
    bool mIgnoreWhitespace;
    bool mResetWhitespace;

    std::vector<BLElementDef*> mElems;
    BLElementDef* mElemPathP;
    int* mElemDistancesP;

    std::vector<BLAttributeDef*> mRequiredAttrs;
    BLAttributeDef* mAttrP;
    unsigned char* mAttrIndexesP;
};

#include "BLDocType_i.h"

#ifdef __BLDOCTYPE_H__
// DO NOT INCLUDE THIS FILE DIRECTLY

class BLAttributeDef
{
public:
    BLAttributeDef(int attrTok, BLAttribute::Type type,
        bool required, BLTokenTable *valueTokTableP)
    {
        mTok(attrTok),
        mType(type),
        mRequired(required),
        mValueTokTableP(valueTokTableP)
    }
};

```



```

    BLAttribute::Type getType()
    int getTok()
    bool isRequired()
    std::vector<int>* getValueToks()
    void addValueTok(int tok)
    bool verifyValue(BLAttribute *attrP);

protected:
    int mTok;
    bool mRequired;
    BLAttribute::Type mType;
    std::vector<int>* mValueToks;
    BLTokenTable *mValueTokTable;
};

inline BLAttributeDef*
BLDocType::getElementDef(int elemTok)
{
    return mElementDefs[elemTok];
}

inline bool
BLDocType::validateElement(BLElement *elemP)
{
    return mElementDefs[elemP->getTok()].validateElement(elemP);
}

inline int
BLDocType::getElementTok(char* str)
{
    return mElementToks.getCaslessToken(str);
}

inline const char*
BLDocType::getElementString(int elemTok)
{
    char* str = mElementStrings[elemTok];
    if (str == NULL)
        return "unknown";
    return str;
}

inline int
BLDocType::getAttributeTok(char* str)
{
    return mAttributeToks.getCaslessToken(str);
}

inline const char*
BLDocType::getAttributeString(int attrTok)
{
    char* str = mAttributeStrings[attrTok];
    if (str == NULL)
        return "unknown";
    return str;
}

inline int
BLDocType::getValueTok(char* str)
{
    return mValueToks.getCaslessToken(str);
}

inline const char*
BLDocType::getValueString(int valueTok)
{
    char* str = mValueStrings[valueTok];
    if (str == NULL)
        return "unknown";
    return str;
}

//BLHtmlDocType.h
#ifndef BLHTMLDOCTYPE_H
#define BLHTMLDOCTYPE_H
#include "BLDocType.h"

class BLHtmlDocType : public BLDocType
{
public:
    enum {
        kCDATAElement = 0,
        kTmlElement,
        kIsotypeElement,
        kItalicElement,
        kBoldElement,
        kUnderlineElement,
        kStrikeElement,
        kBigElement,
        kSmallElement,
        kSubscriptElement,
        kSuperscriptElement,
        kEmphasisElement,
        kStrongElement,
        kFontElement,
        kBaseFontElement,
        kLinebreakElement,
        kBodyElement,
        kDivElement,
        kAnchorElement,
        kCenterElement,
        kMapElement,
        kAreaElement,
        kImageElement,
        kHorizontalRuleElement,
        kParagraphElement,
        kHeading1Element,
        kHeading2Element,
        kHeading3Element,
        kHeading4Element,
        kHeading5Element,
        kHeading6Element,
        kPreElement,
        kBlockQuoteElement,
        kDefinitionListElement,
        kDefinitionTermElement,
        kDefinitionDefinitionElement,
        kOrderedListElement,
        kUnorderedListElement,
        kListElement,
        kFormElement,
        kInputElement,
        kSelectElement,
        kOptionElement,
        kTextAreaElement,
        kTableElement,
        kTableRowElement,
        kTableHeaderElement,
        kTableDataElement,
        kCaptionElement,
        kHeadElement,
        kTitleElement,
        kBaseElement,
        kMetaElement,
        kStyleElement,
        kScriptElement,
        kNumElements
    };

    enum {
        kSizeAttr = 0,
        kColorAttr,
        kBackgroundColorAttr,
        kTextAttr,
    };
};

```

```

kLinkAttr,
kVisitedLinkAttr,
kActiveLinkAttr,
kAlignAttr,
kNameAttr,
kHrefAttr,
kUseMapAttr,
kIsMapAttr,
kNoShadeAttr,
kNoHrefAttr,
kShapeAttr,
kCoordsAttr,
kAltAttr,
kSrcAttr,
kHeightAttr,
kWidthAttr,
kBorderAttr,
kHorizontalSpaceAttr,
kVerticalSpaceAttr,
kStartAttr,
kActionAttr,
kMethodAttr,
kEncodingTypeAttr,
kTypeAttr,
kValueAttr,
kCheckedAttr,
kMaxLengthAttr,
kMultipleAttr,
kSelectedAttr,
kRowsAttr,
kColsAttr,
kCellSpacingAttr,
kCellPaddingAttr,
kHorizontalAlignAttr,
kVerticalAlignAttr,
kRowspanAttr,
kColspanAttr,
kHttpEquivAttr,
kContentAttr,
kNumAttrs

};

enum {
    kLeftValue = 0,
    kCenterValue,
    kRightValue,
    kRectValue,
    kCircleValue,
    kPolyValue,
    kTopValue,
    kMiddleValue,
    kBottomValue,
    kGetValue,
    kPostValue,
    kTextValue,
    kPasswordValue,
    kCheckboxValue,
    kRadioValue,
    kSubmitValue,
    kResetValue,
    kFileValue,
    kHiddenValue,
    kImageValue,
    kRefreshValue,
    kNumValues
};

BLHtmlDocType();

virtual int initialize();
virtual const char* getDocTypeString();
virtual BLElementDef* getRootElemDef();

protected:
void initClasses();

void initElements();
void initElementToks();
void initAttrToks();
void initValueToks();
void initEntityToks();
};

#ifdef _BLHTMLDOC_TYPE_H_
// BLHtmlParser.h
#endif
#ifdef _BLHTMLPARSER_H_
// BLHtmlParser.h
#include "BLDocStream.h"
#include "utility"

#define kParseBufSize 4096
#define kEntityBufSize 10
#define kMaxEntitySize 7
#define kMaxTableColumns 50

class BLHtmlParser : public BLTransportStream
{
public:
    BLHtmlParser(BLDocType *docTypeP, BLDocStream *docStreamP);

    // call before parsing new document
    virtual int init(int length, BLTransformManager::ContentType type);

    // set final = true when this is the last chunk of the doc
    // (size may be 0)
    virtual int put(char *data,
                    int size,
                    bool final);

protected:
    void parseBuf(char *data,
                  int size,
                  bool final);

    inline void doLinebreaks();
    inline void beginTag();
    inline void beginTagDone();
    inline void endTag();
    inline void attribute();
    inline void value();
    inline void charData();
    inline void doEntity(int entity);
    inline void charEntity();
    inline void decimalEntity();
    inline void hexEntity();
    inline void beginDocument();
    inline void endDocument();
    inline int lookahead(char closeChar,
                        int minDistance,
                        int maxDistance,
                        char *buf,
                        int buflen);

    void dumpScratch();

    int mState;
    bool mLastCharDataSpace;
    bool mLastValuesSpace;

    char mParseBuf[kParseBufSize];
    char mScratchBuf[kParseBufSize];
    char mTitleBuf[kTitleSize];

    BLElement mElement;
    BLAttribute mAttribute;

    int mScratchDistance;
    int mScratchMinDistance;
    char mScratchCloseChar;
    char mLookaheadChar;
};

```

```

char mScratchChar;

char *mParseP;
char *mValueP;
char *mScratchP;

BLDocType* mDocTypeP;
BLDocStream* mDocStreamP;

//
#endif /* __BLHTMLPARSER_H_ */ /// BLTokenTable.h

#ifdef __BLTOKENTABLE__
#define __BLTOKENTABLE__
#include "ace/Hash_Map_Manager.h"
#include "BLUtils.h"

#define kMaxTokenSize 256

class BLTokenTable : public ACE_Hash_Map_Manager<const char*, int, ACE_Null_Mutex>
{
public:
    BLTokenTable();
    ~BLTokenTable();

    inline int getToken(char *str);
    inline int getCaselessToken(char *str); //note: alters original token
    void bindToken(char *str, int token);
    void bindCaselessToken(char *str, int token);

};

inline int
BLTokenTable::getToken(char *str)
{
    ACE_Hash_Map_Entry<const char*, int> *entry;
    if (find(str, entry) == 0)
        return entry->int_id;
    return -1;
}

inline int
BLTokenTable::getCaselessToken(char *str)
{
    char buf[kMaxTokenSize];
    int i;
    for (i=0; i < kMaxTokenSize - 1; i++)
    {
        if (str[i] == '\0')
            break;
        buf[i] = BLUtils::caseless(str[i]);
    }
    buf[i] = '\0';
    return getToken(buf);
}

inline void
BLTokenTable::bindToken(char *str, int token)
{
    bind(BLUtils::strdup(str), token);
}

inline void
BLTokenTable::bindCaselessToken(char *str, int token)
{
    int len = ACE_OS::strlen(str);
    char *newstr = new char[len + 1];

```

```

for (int i=0; i<len; i++)
    newstr[i] = BLUtils::caseless(str[i]);
newstr[len] = '\0';

bind(newstr, token);
}

#endif /* __BLTOKENTABLE__ */ /// BLTransformManager.h

#ifdef __BLTRANSFORMMANAGER__
#define __BLTRANSFORMMANAGER__
#include <vector>
#include <utility>

// prevent overly long symbols
#define BLTransformManager BLTM
#define BLTransformationFactory BLTF

class ACE_Null_Mutex;

template <class EXT_ID, class INT_ID, class ACE_LOCK>
class ACE_Hash_Map_Manager;

class BLTransformation;
class BLTransformationFactory;
class BLClientInfo;
class BLTransportStream;
class BLTransformManager
{
public:
    typedef enum {
        kUnknownType = -1,
        kHTMLType = 0,
        kXMLType,
        kWMMLType,
        kJPEGType,
        kGIFType,
        kPNGType,
        kWBMPType,
        kPalmBMFType,
        kNumOfEncodedTypes,
        kNumContentTypes
    } ContentType;

    static BLTransformManager* instance();

    int initialize();
    void finalize();

    // returns NULL on error
    BLTransformation* getTransformation(BLClientInfo *info,
        ContentType fromType,
        BLTransportStream *streamP);

    BLTransformation* getTransformation(BLClientInfo *info,
        char* fromType,
        BLTransportStream *streamP);

    ContentType stringToContentType(char *str);
    const char* contentTypeOfString(ContentType type);

    void registerTransformation(ContentType fromType, ContentType toType,
        BLTransformationFactory *factoryP);

protected:
    BLTransformManager();
    ~BLTransformManager();

    void setContentTypeString(ContentType type, char *str);

    static BLTransformManager* sInstanceP;
    static BLTransformationFactory* sFactoryDefns[];

```



```

enum {
    kDataElement = 0,
    kXmlElement,
    kCardElement,
    kDoElement,
    kCommandEvent,
    kHeadElement,
    kTemplateElement,
    kAccessElement,
    kMetaElement,
    kCoElement,
    kPrevElement,
    kRefreshElement,
    kNoopElement,
    kPostfieldElement,
    kSetVarElement,
    kSelectElement,
    kOptGroupElement,
    kOptionElement,
    kInputElement,
    kFieldSetElement,
    kTimerElement,
    kImageElement,
    kAnchorElement,
    kAElement,
    kTableElement,
    kTableRowElement,
    kTableDataElement,
    kEmphasisElement,
    kStrongElement,
    kBoldElement,
    kItalicElement,
    kUnderlineElement,
    kRigidElement,
    kSmallElement,
    kParagraphElement,
    kLinebreakElement,
    kPreElement,
    kNumElements
};

enum {
    kRefAttr = 0,
    kNameAttr,
    kValueAttr,
    kAlignAttr,
    kSrcAttr,
    kLocalSrcAttr,
    kAltAttr,
    kVSpaceAttr,
    kHSpaceAttr,
    kHeightAttr,
    kWidthAttr,
    kTitleAttr,
    kNewContextAttr,
    kOrderedAttr,
    kOnEnterForwardAttr,
    kOnEnterBackwardAttr,
    kOnTimerAttr,
    kCDATAAttr,
    kEntityAttr,
    kTypeAttr,
    kDomainAttr,
    kPathAttr,
    kHttp-equivAttr,
    kForuaAttr,
    kContentAttr,
    kSchemeAttr,
    kSendRefererAttr,
    kMethodAttr,
    kEncTypeAttr,
    kAcceptCharsetAttribute,
    kNameAttr,
    kValueAttr,
    kMultipleAttr,

    kTabIndexAttr,
    kOnPickAttr,
    kFormAttr,
    kEmptyAttr,
    kSizeAttr,
    kMaxLengthAttr,
    kAccessKeyAttr,
    kColumnsAttr,
    kLabelAttr,
    kOptionalAttr,
    kModeAttr,
    kIDAttr,
    kNumAttrs

};

enum {
    kLeftValue = 0,
    kCenterValue,
    kRightValue,
    kTopValue,
    kMiddleValue,
    kBottomValue,
    kTrueValue,
    kFalseValue,
    kPostValue,
    kGetValue,
    kSetValue,
    kPasswordValue,
    kWrapValue,
    kNowrapValue,
    kAcceptValue,
    kDeleteValue,
    kHelpValue,
    kOnPickValue,
    kOnEnterBackwardValue,
    kOnEnterForwardValue,
    kOnTimerValue,
    kOptionsValue,
    kPrevValue,
    kResetValue,
    kUnknownValue,
    kNumValues
};

BLMmlDocType();

virtual int initialize();
virtual const char* getDocTypeString();
virtual BLMmlElementDef* getRootElementDef();

protected:
    void initClasses();
    void initElements();
    void initElementToks();
    void initAttributeToks();
    void initValueToks();
    void initEntityToks();
};

#ifdef __BLMMLDOCTYPE_H__
#endif
#ifdef __BLMMLTABLE_H__
#endif
#include "BLMxmlOutput.h"

class BLMmlTable : public BLMxmlTable
{
public:
    BLMmlTable() : BLMxmlTable() {}

    int initialize();

protected:
    int makeElementTable();
};

```

```

int makeValueTable();
int makeAttributeTable();
};

#ifdef __BLXMLTABLE_H__
// BLXMLOutput.h

#include "BLXMLOUTPUT_H"
#define __BLXMLOUTPUT_H__

#include "BLTransportWriter.h"
#include "BLTransportStream.h"
#include "BLDocStream.h"

class BLXMLOutput : public BLDocStream
{
public:
    BLXMLOutput(BLTransportStream *transportP, BLTransformManager::ContentType type);

    virtual int beginDoc(BLDocType *docTypeP);
    virtual int endDoc();

    virtual int beginElement(BLDocType *elemP);
    virtual int endElement(int elemTok);
    virtual int charData(const char *data, int length);

protected:
    void charData(const char *str);
    void writeChar(char c);

    BLDocType *mDocTypeP;
    BLTransportStream *mTransportP;
    BLTransportWriter *mStream;
    BLTransformManager::ContentType mContentType;
};

#ifdef __BLXMLOUTPUT_H__
// BLXMLParser.h

#include "BLDocType.h"
#include "BLTransportStream.h"
#include "BLTransportWriter.h"
#include "BLDocStream.h"
#include "BLXMLParser.h"

class BLXMLParser : public BLTransportStream
{
public:
    BLXMLParser(BLDocType *docTypeP, BLDocStream *docStreamP);
    ~BLXMLParser();

    virtual int init(int length, BLTransformManager::ContentType type);
    virtual int put(char *data,
                    int size,
                    bool final);

protected:
    static void xmlStartElementHandler( void *userData,
                                        const char *name,
                                        const char **atts);

    static void xmlEndElementHandler(void *userData,
                                     const char *name);
    static void xmlCDATAHandler(void *userData,
                                const char *s,
                                int len);

    void error(const char *message, int lineNumber);

    BLDocType *mDocTypeP;
    BLDocStream *mDocStreamP;

    bool mFastInitialWhitespace;
    bool mPreserveWhitespace;
    bool mIgnoreWhitespace;
    bool mLastWhitespace;

```

## Browser Source Code

## Browser Source Code

```
// BLApplication.cpp

/** general application methods.
 */

#include "BLApplication.h"
#include "Bliterator.h"

BLApplication *
BLApplication::mThis = NULL;

BLApplication::BLApplication()
{
}

BLApplication::~BLApplication()
{
}

void
BLApplication::initialize()
{
    mPrefs = NULL;
    mCurrentFormHandler = NULL;
    mCurrentMenuHandler = NULL;
    mThis = this;
}

/*
Launch codes (from Palm devzone knowledge base):

Execute the specified Network login script plugin command.
Provide information about the commands that your Network
script plugin executes.
Add a record to a database.
Schedule next alarm or perform quick actions such as sounding
alarm tones.
Respond to country change.
Display specified alarm dialog or perform time-consuming
alarm-related actions.
Let application override display of dialog asking user if
incoming data via the exchange manager.
Notify application that it should receive incoming data via
the exchange manager.
Find a text string.
Go to a particular record, display it, and optionally select
the specified text.
Launch Clipper application and open a URL. (Palm VII system
only.)
Initialize database.
Look up data. In contrast to sysAppLaunchCmdFind, a level of
For example, look up a phone number associated with a name.
Launch normally.
Launch application and open a database. (Palm VII system
only.)
sysAppLaunchCmdCalledFromApp Tell preferences panel that it was invoked from an
application, not the Preferences application.
sysAppLaunchCmdReturnFromPanel Tell an application that it's restarting after preferences
panel has been called.
sysAppLaunchCmdSaveData Notify applications that a HotSync has been completed.
sysAppLaunchCmdSyncNotify Sent to the Security application to request that the system
be locked down.
sysAppLaunchCmdSystemReset Respond to system reset. No UI is allowed during this launch
code.
sysAppLaunchCmdTimeChange Respond to system time change.
sysAppLaunchCmdURLParams Launch an application with parameters from Clipper. (Palm VII
system only.)
*/

void BLApplication::getEvent(EventType & event)
{
    #ifdef ENABLE_NETWORKING
    mNetwork.getEvent(event);
    #else
    EvtGetEvent(&event, evtWaitForever);
    #endif
}

DWORD
BLApplication::eventLoop()
{
    Word error = 0;
    EventType event;
    do {
        getEvent(event);
        if (!SysHandleEvent(&event))
            if (!MenuHandleEvent(0, &event, &error))
                if (!HandleEvent(&event))
                    FrmDispatchEvent(&event);
    } while (event.eType != appStopEvent);
    if (mCurrentFormHandler)
        mCurrentFormHandler->formClose();
    return error;
}

Boolean
BLApplication::handleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr formP;
    Word formID;
    switch(eventP->eType)
    {
        case frmLoadEvent:
            formID = eventP->data.frmLoad.formID;
            formP = FrmInitForm(formID);
            FrmSetActiveForm(formP);
            FrmSetEventHandler(formP, formEventDispatcher);
            setFormHandler(formID);
            mCurrentFormHandler->formLoad(formP);
            handled = true;
            break;
        case frmOpenEvent:
            formP = mCurrentFormHandler->getFormPtr();
            FrmDrawForm(formP);
            mCurrentFormHandler->formOpen();
            handled = true;
            break;
        case frmCloseEvent:
            formP = mCurrentFormHandler->getFormPtr();
            mCurrentFormHandler->formClose();
            FrmEraseForm(formP);
            FrmDeleteForm(formP);
    }
}
```



```

        handled = true;
        break;
    }

    case menuEvent:
        if(mCurrentMenuHandler)
            handled = mCurrentMenuHandler->handleEvent(eventP);
        break;
    }

    #ifdef ENABLE_NETWORKING
        case inetSocketReadyEvent:
            if(((INetEventType *)eventP)->data.inetSocketReady.inputReady)
            {
                Handle sockH = ((INetEventType *)eventP)->data.inetSocketReady.sockH;
                mNetwork.dataReady(sockH);
            }
            break;

        case inetSocketStatusChangeEvent:
            // Handle sockH2 = ((INetEventType *)eventP)->data.inetSocketStatusChange.sockH;
            mNetwork.displayStatus(((INetEventType *)eventP)->data.inetSocketStatusChange.status);
            break;
        #endif
    }

    return handled;
}

Boolean
BLApplication::formEventDispatcher(EventPtr eventP)
{
    Boolean handled = false;

    if(mThis->mCurrentFormHandler)
        handled = mThis->mCurrentFormHandler->handleEvent(eventP);

    return handled;
}

void
BLApplication::registerHandler(BLEventHandler * handler)
{
    mHandlers.add(handler);
}

void
BLApplication::setFormHandler(Word formID)
{
    BLIterator<BLEventHandler*, kHandlerInc> iter(mHandlers);
    while(iter.hasNext())
    {
        BLEventHandler* handler = iter.getNext();
        if( handler->doesHandlerType(BLEventHandler::kFormType) &&
            handler->doesHandleObject(formID))
            mCurrentFormHandler = dynamic_cast<BLForm *>(handler);
    }
}

#ifdef ENABLE_NETWORKING
void
BLApplication::startNetworking()
{
    mNetwork.start();
}
}

// This constructor should be called in the app start call
mBookmarkDBP = new BLBookmarkDB();
mBookmarkDBP->openBookmarkDB();
mChoicesHandle = NULL;
mChoicesPtrsHandle = NULL;
mLastOpenUrlP = NULL;
}

BLBookmark::BLBookmark ()
{
    // This constructor should be called in the app start call
    mBookmarkDBP = new BLBookmarkDB();
    mBookmarkDBP->openBookmarkDB();
    mChoicesHandle = NULL;
    mChoicesPtrsHandle = NULL;
    mLastOpenUrlP = NULL;
}
}

```

```

void
BLApplication::stopNetworking()
{
    mNetwork.stop();
}

#ifdef ENABLE_NETWORKING
// BLApplication.h
#pragma once

#include "BLVector.h"
#include "BLForm.h"
#include "BLNetworking.h"

class BLPrefs;

class BLApplication
{
public:
    enum { appFileCreator = '????',
           ourMinVersion = sysMakeROMVersion(2,0,0,sysROMStageRelease,0),
           kHandlerInc = 4 };

    virtual
    BLApplication();
    ~BLApplication();

    virtual
    initialize();

    virtual
    launchNormal() = 0;

    inline
    void
    DWord
    Boolean
    static
    Boolean
    void
    BLPrefs *
    void
    void
    void

    getEvent(EventType & event);
    eventLoop();
    handleEvent(EventPtr eventP);
    formEventDispatcher(EventPtr eventP);
    setPrefs(BLPrefs * prefs) { mPrefs = prefs; }
    getPrefs() { return mPrefs; }
    registerHandler(BLEventHandler * handler);
    setFormHandler(Word formID);

protected:
    #ifdef ENABLE_NETWORKING
        void
        stopNetworking();
        BLNetworking mNetwork;
    #endif

private:
    static
    BLApplication * mThis;

    BLVector<BLEventHandler*, kHandlerInc> mHandlers;
    BLPrefs * mPrefs;

    BLForm * mCurrentFormHandler;
    BLEventHandler * mCurrentMenuHandler;
};

#include "BLBookmark.h"

BLBookmark::BLBookmark ()
{
    // This constructor should be called in the app start call
    mBookmarkDBP = new BLBookmarkDB();
    mBookmarkDBP->openBookmarkDB();
    mChoicesHandle = NULL;
    mChoicesPtrsHandle = NULL;
    mLastOpenUrlP = NULL;
}
}

```

```

mBookmarkDBP->loadBookmarkRecord(bookmarksDetailsOfflineCheckBox);
BookmarksDetailsURLField, BookmarksDetailsOfflineCheckBox);
FrmDialog(frmP);
FrmDeleteForm(frmP);
FrmSetActiveForm(previousForm);
handled = true;
}
else
{
    // if no bookmarks, issue alert
    FrmAlert(BookmarksDetailsAlert);
    handled = true;
}
}
else if (eventP->data.ctrlEnter.controlID == BookmarksEditDeleteButton)
{
    // if the current record is valid
    if (mBookmarkDBP->isValidRecord())
    {
        if (mBookmarkDBP->getNumRecords() > 0) {
            mBookmarkDBP->deleteCurrentRecord();
            numRecords = mBookmarkDBP->getNumRecords();
        }
        frmP = FrmGetActiveForm();
        lstP = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,
            BookmarksEditListList));
        LstSetListChoices(lstP, NULL, numRecords);
        // after delete set selection to nothing
        LstSetSelection(lstP, -1);
        LstDrawList(lstP);
    }
    else
    {
        FrmAlert(BookmarksDeleteAlert);
    }
    handled = true;
}
}
break;
}
case menuEvent:
    // First clear the menu status from the display.
    MenuEraseStatus(0);
    // process menu commands for the bookmark form
    bookmarkDoMenuCommand(eventP->data.menu.itemID);
    handled = true;
    break;
}
case lstSelectEvent:
    // Get the current selection from the list and set the current record to edit.
    frmP = FrmGetActiveForm();
    lstP = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,
        BookmarksEditListList));
    mBookmarkDBP->setCurrentRecord(LstGetSelection(lstP));
    handled = true;
    break;
}
case frmUpdateEvent:
    if (eventP->data.frmUpdate.updateCode == kListUpdateCode)
        initBookmarkEditForm(BookmarksEditListList);
    handled = true;
    break;
}
case frmCloseEvent:
    break;
}
return(handled);
}
}
Boolean
BLBookmark::bookmarksAddHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    switch (eventP->eType)
    {
        case ctrlSelectEvent:
            if (eventP->data.ctrlEnter.controlID == GotoPageGoButton)
            {
                // get the url and pass it to the browser
                mOpenUrlP = FldGetTextPtr((FldType*)BLUtils::getObjectPtr
                    (GotoPageURLField));
                mLastOpenUrlP = BLUtils::cloneString(mOpenUrlP);
            }
            break;
        case menuEvent:
            // First clear the menu status from the display.
            MenuEraseStatus(0);
            BLBookmark::bookmarkDoMenuCommand(eventP->data.menu.itemID);
            handled = true;
            break;
        case keyDownEvent:
            handled = processKeyDown(eventP);
            break;
        case penDownEvent:
            testInsPos(eventP);
            break;
    }
    return(handled);
}
Boolean
BLBookmark::bookmarksEditHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    Boolean isCreated = true;
    FormPtr frmP, previousForm;
    UInt numRecords;
    switch (eventP->eType)
    {
        case ctrlSelectEvent:
            if (eventP->data.ctrlEnter.controlID == BookmarksEditNewButton)
            {
                previousForm = FrmGetActiveForm();
                BLUtils::initBasicDialogForm(frmP, BookmarksAddForm, bookmarksAddHandleEvent,
                    BookmarksAddNameField);
                BLUtils::setFieldToStrResource(BookmarksAddURLField, DefaultProtocolString);
                FrmDialog(frmP);
                FrmDeleteForm(frmP);
                FrmSetActiveForm(previousForm);
                handled = true;
            }
            else if (eventP->data.ctrlEnter.controlID == BookmarksEditDetailsButton)
            {
                if (mBookmarkDBP->isValidRecord())
                {
                    previousForm = FrmGetActiveForm();
                    BLUtils::initBasicDialogForm(frmP, BookmarksDetailsForm,
                        bookmarksDetailsHandleEvent, BookmarksDetailsNameField);
                }
            }
    }
}

```

```

if (numRecords)
{
    // Get a pointer to the list object.
    lstp = (ListPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, listID));

    // Set the number of list choices.
    ListSetListChoices(lstP, NULL, numRecords);

    // Set the callback function to draw each list entry.
    ListSetDrawFunction(lstP, BLBookmarkDB::editFormListDrawItem);

    ListSetSelection(lstP, mBookmarkDBP->getCurrentRecord());

}
FrmDrawForm(frmP);
}

/*.....*/
* FUNCTION:      updateBookmarksPopup
* DESCRIPTION:   sets the updated database info for the given pop up list
* PARAMETERS:   frm - pointer to the main form.
* RETURNED:     Nothing.
*.....*/
void
BLBookmark::updateBookmarksPopup(Word popupListID)
{
    FrmPtr      frmP;
    ListPtr     lstP;
    UInt        numRecords;
    CharPtr     choicesP;
    RectangleType listRect;
    Int         itemIndex, lstWidth;

    freeMemForBookmarksPopup();
    frmP = FrmGetActiveForm();

    // Get the number of records currently in the application's database.
    numRecords = mBookmarkDBP->getNumRecords();

    // Get a pointer to the list object.
    itemIndex = FrmGetObjectIndex(frmP, popupListID);

    // Get a pointer to the list object.
    lstP = (ListPtr) FrmGetObjectPtr(frmP, itemIndex);

    // Get the usable width of the list rectangle.
    FrmGetObjectBounds(frmP, itemIndex, &listRect);
    lstWidth = listRect.extent.x - 2;

    // Allocate an initial block for the list choices.
    mChoicesHandle = MemHandleNew(sizeof(char));

    choicesP = (CharPtr) MemHandleLock(mChoicesHandle);
    *choicesP = 0;

    // the database updates the choices
    mBookmarkDBP->constructPopupListArray(itemIndex, numRecords, lstWidth, mChoicesHandle,
    choicesP);

    // Note that we add one to the numRecords to include the Edit Bookmarks command
    // at the bottom of the popup list.

    // Create an array of pointers from the choices strings.
    mChoicesPtrsHandle = SysFormPointerArrayToStrings(choicesP, numRecords + 1);

    // Set the list choices from the array of pointers.
    ListSetListChoices(lstP, (Char**) MemHandleLock(mChoicesPtrsHandle), numRecords + 1);

    // Set the number of visible items
    ListSetHeight(lstP, numRecords + 1);
}

/*.....*/
* FUNCTION:      freeMemForBookmarksPopup
* DESCRIPTION:   frees the memory assoc with the popup list. Should be
*               between calls to updatePopupList and when app quits.
* PARAMETERS:     Nothing.
* RETURNED:       Nothing.
*.....*/
void
BLBookmark::freeMemForBookmarksPopup(void)
{
    // Free the memory blocks allocated for the list.
    if (mChoicesHandle)
    {
        MemHandleFree(mChoicesHandle);
        mChoicesHandle = NULL;
        MemHandleFree(mChoicesPtrsHandle);
        mChoicesPtrsHandle = NULL;
    }
}

/* AUTO COMPLETE CODE */
void
BLBookmark::initGotoPageDialog(void)
{
    CharPtr string = mLastOpenUrlP;

    if (string == NULL)
        string = "http://";

    setCurrInsPos(strlen(string) - 1);
    setInsNewInsPos(false);
    mOpenUrlP = NULL;
}

void
BLBookmark::setCurrInsPos(int i)
{
    mCurrInsPos = i;
}

void
BLBookmark::setInsNewInsPos(Boolean val)
{
    mIsNewInsPos = val;
}

// if user clicked in field set mIsNewInsPos to true
void
BLBookmark::testInsPos(EventPtr eventP)
{
    RectangleType rectF;
    PointType ptr;
    FieldPtr fldP;

    fldP = (FieldPtr) BUUtils::getObjectPtr (GotoPageURLField);
    FieldGetBounds (fldP, rectF);
    ptr = eventP->data.penUp.start;
    if (RectPtInRectangle (ptr.x, ptr.y, rectF))
    {
        mIsNewInsPos = true;
    }
}

```

```

static BLBookmarkDB * mBookmarkDBP;

```

```

class BLBookmark
{
public:
    BLBookmark();
    ~BLBookmark();

    // Bookmark Form Handlers
    Boolean bookmarkPopupHandleEvent(EventPtr eventP);
    Boolean gotoPageHandleEvent(EventPtr eventP);
    Boolean bookmarksEditHandleEvent(EventPtr eventP);

    // Callbacks
    static Boolean bookmarkAddHandleEvent(EventPtr eventP);
    static Boolean bookmarkDetailsHandleEvent(EventPtr eventP);
    void updateBookmarksPopup(Word popupListID);
    static void bookmarkDoMenuCommand(Word command);
    void initBookmarkEditForm(Word listID);

    // Handles
    void Handle mChoicesHandle, mChoicesPtrsHandle; // handles for dynamic list data

    // auto complete
    void initGotoPageDialog();
    void setCurrentPos(int i);
    void setNewPos(Boolean val);

    CharPtr getUrl(Word index);
    CharPtr getOpenUrl() { return mOpenUrl; }

    CharPtr getLastOpenUrl() { return mLastOpenUrl; }

protected:

    // Helper methods
    FieldPtr getFocusObjectPtr(void);
    void freeMemForBookmarksPopup(void);
    Boolean processKeyDown(EventPtr eventP);
    void testInsPos(EventPtr eventP);

    // Member Variables
    enum { kListUpdateCode = 0x01 };

    // Auto complete vars
    Word mCurrInsPos, mNewInsPos;
    CharPtr mOpenUrl;
    CharPtr mLastOpenUrl;

}; #include "BLBookmarkDB.h"

const CharPtr kUntitledBookmarkNameP = "Untitled-";

BLBookmarkDB::BLBookmarkDB()
{
    mBookmarkAppType = 'Book'; // type for application. must be 4 chars, mixed
    case. mBookmarkDBType = 'Book'; // type for application database. must be 4 chars,
    mixed case. mBookmarkDBName = "BookmarkDB"; // name for application database. up to 31
    characters.
    mCurrentRecord = kNoRecordSelected; // Set current bookmark database record to no
    current record.
    kEditBookmarksCommand = "Edit Bookmarks...";
}

BLBookmarkDB::~BLBookmarkDB()
{
    closeBookmarkDB();
}

```

```

Boolean
BLBookmark::processKeyDown(EventPtr eventP)
{
    CharPtr newCharP, prevUrlP, newUrlP, urlSuggP;
    int prevUrlLen, newUrlLen, suggestionUrlLen;
    FieldPtr fldP;
    char newChar;
    Boolean handled = false;
    if (mNewInsPos) {
        setNewInsPos(false);
        return false;
    }
    // get the new character entered
    newChar = (char)eventP->data.keyDown.chr;
    newCharP = &newChar;
    fldP = (FieldPtr) BLUtils::getObjectPtr (GotoPageURLField);
    // get the prev url
    prevUrlP = fldP->getTextPtr(fldP);
    prevUrlLen = strlen(prevUrlP);
    mCurrInsPos = fldP->getPosition (fldP);
    // check if the user has deleted a letter
    if (newChar == '\b')
    {
        newUrlLen = prevUrlLen - 1;
        mCurrInsPos--;
    }
    else
    {
        newUrlLen = prevUrlLen + 1;
        mCurrInsPos++;
    }
    // only do the auto complete if the user has added a letter
    if (newUrlLen > prevUrlLen)
    {
        newUrlP = new Char[newUrlLen + 1];
        strcpy(newUrlP, prevUrlP);
        // tack on the new character to the url
        strcat(newUrlP, newCharP);
        urlSuggP = mBookmarkDBP->getFirstUrlMatch(newUrlP);
        if (urlSuggP)
        {
            BLUtils::writeToStrField(GotoPageURLField, urlSuggP);
            suggestionUrlLen = strlen(urlSuggP);
            fldP->setSelection (fldP, mCurrInsPos, suggestionUrlLen);
            fldP->setPosition (fldP, mCurrInsPos);
            delete urlSuggP;
            handled = true;
        }
        else handled = false;
        delete newUrlP;
    }
    return handled;
}

CharPtr
BLBookmark::getUrl(Word index)
{
    CharPtr urlP;
    mBookmarkDBP->getUrlForRecord(index, urlP);
    return urlP;
} #pragma once
#include "BLBookmarkDB.h"

```

```

    error = DmSetDatabaseInfo(0, dbID, NULL, &dbAttrs, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
    // Check for fatal error.
    ErrFatalDisplayIf(error, "Could not set bookmark database info.");
}

void BLBookmarkDB::setCurrentRecord(void)
{
    return mCurrentRecord;
}

void BLBookmarkDB::setCurrentRecord(Word index)
{
    mCurrentRecord = index;
}

UInt BLBookmarkDB::getNumRecords(void)
{
    return DmNumRecords(mBookmarkDB);
}

Boolean BLBookmarkDB::IsValidRecord(void)
{
    if (mCurrentRecord == kNoRecordSelected) return false;
    return true;
}

/* ***** */
* FUNCTION: openBookmarkDB
* DESCRIPTION: This routine opens the bookmark database and sets globals.
* PARAMETERS: None.
* RETURNED: nothing.
* ***** */
void BLBookmarkDB::openBookmarkDB(void)
{
    Err error;
    UInt cardNo;
    LocalID dbID;
    UInt dbAttrs;
    UInt mode;

    // Find the bookmark database.
    mode = dmModeReadWrite;
    mBookmarkDB = DmOpenDatabaseByTypeCreator(mBookmarkDBType, mBookmarkAppType, mode);
    if (!mBookmarkDB)
    {
        // The database doesn't exist, create it now.
        error = DmCreateDatabase(0, mBookmarkDBName, mBookmarkAppType, mBookmarkDBType, false);

        // Check for fatal error.
        ErrFatalDisplayIf(error, "Could not create new bookmark database.");

        // Find the application's database.
        mBookmarkDB = DmOpenDatabaseByTypeCreator(mBookmarkDBType, mBookmarkAppType, mode);

        // Get info about the database
        error = DmOpenDatabaseInfo(mBookmarkDB, &dbID, NULL, NULL, &cardNo, NULL);

        // Check for fatal error.
        ErrFatalDisplayIf(error, "Could not get bookmark database info.");

        // Get attributes for the database
        error = DmDatabaseInfo(0, dbID, NULL, &dbAttrs, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

        // Check for fatal error.
        ErrFatalDisplayIf(error, "Could not get bookmark database info.");

        // Set the new attributes in the database
    }
}

```

```

    error = DmSetDatabaseInfo(0, dbID, NULL, &dbAttrs, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
    // Check for fatal error.
    ErrFatalDisplayIf(error, "Could not set bookmark database info.");
}

void BLBookmarkDB::closeBookmarkDB(void)
{
    DmCloseDatabase(mBookmarkDB);
}

/* ***** */
* FUNCTION: saveBookmarkRecord
* DESCRIPTION: This function writes the bookmark data into the database.
* It checks for the case where there is no user input
* PARAMETERS: None.
* RETURNED: nothing.
* ***** */
void BLBookmarkDB::saveBookmarkRecord (Word nameFieldID, Word urlFieldID, Boolean isNewRecord, Boolean isCached)
{
    VoidHand recordH;
    VoidPtr recordP;
    UInt nameSize=0;
    UInt nameLen=0;
    UInt urlSize=0;
    UInt urlLen=0;
    Char zero=0;
    UInt index=0;
    Char nameP;
    CharPtr urlP;

    nameP = fldGetTextPtr((FieldType*)BLUtils::getObjectPtr(nameFieldID));
    urlP = fldGetTextPtr((FieldType*)BLUtils::getObjectPtr(urlFieldID));

    // Compute the size of the bookmark record.
    if (nameP) {
        nameLen = StrLen(nameP);
        if (nameLen > 0) {
            nameSize = nameLen + 1;
        }
        else {
            nameSize = StrLen(kUntitledBookmarkNameP) + 1;
        }
    }
    else {
        nameLen = 0;
        nameSize = StrLen(kUntitledBookmarkNameP) + 1;
    }

    if (urlP)
        urlSize = StrLen(urlP) + 1;
    else
        urlSize = 1;

    if (!isNewRecord)
    {
        // Allocate a chunk in the database for the new record.
        recordH = DmNewRecord(mBookmarkDB, &index, (nameSize + urlSize + sizeof(Boolean)));
        mCurrentRecord = index;
    }
    else
    {
        if (mCurrentRecord != kNoRecordSelected && DmNumRecords(mBookmarkDB) > 0)
        {

```

```

* RETURNED: nothing
void BLBookmarkDB::loadBookmarkRecord (Word nameFieldID, Word urlFieldID, Word checkBoxID)
{
    Word offset = 0;
    VoidHand recordH;
    CharPtr recordP;

    // Check to make sure there is a record to retrieve.
    if (mCurrentRecord != kNoRecordSelected && mNumRecords(mBookmarkDB) > 0)
    {
        // Get a handle for the current record.
        recordH = DmGetRecord (mBookmarkDB, mCurrentRecord);
        // Lock down the handle and get a pointer to the record data.
        recordP = (CharPtr)MemHandleLock(recordH);
        BLUtils::writeStrToField(nameFieldID, recordP);

        // Now do the same for the url field
        recordP += StrLen(recordP) + 1;
        BLUtils::writeStrToField(urlFieldID, recordP);

        // advance pointer to Boolean isCached value
        recordP += StrLen(recordP) + 1;
        CtIsSetValue((ControlPtr)BLUtils::getObjectPtr(checkBoxID), recordP(0));

        // Unlock the database record.
        MemHandleUnlock(recordH);

        // Release the record to the database system.
        // (The zero means that the record data hasn't been changed.)
        DmReleaseRecord(mBookmarkDB, mCurrentRecord, 0);
    }
}

/*****
* FUNCTION: getUrlForRecord
* DESCRIPTION: This function copies the url for a given record index
*               Note that the caller should FREE the memory after it is done.
*               Also a test for no an empty url should be considered.
* PARAMETERS: index - of bookmark
*               urlP
* RETURNED: Nothing.
*****/
void BLBookmarkDB::getUrlForRecord(Word index, CharPtr& urlP)
{
    VoidHand recordH;
    CharPtr recordP;
    Word numRecords;

    numRecords = getNumRecords();
    if((index < 0) || (index >= numRecords))
    {
        ErrDisplay("Attempted to get url for record index out of range");
    }
    else
    {
        // Get a handle for the current record.
        recordH = DmGetRecord (mBookmarkDB, index);
        // Lock down the handle and get a pointer to the record data.
        recordP = (CharPtr)MemHandleLock(recordH);
        // advance ptr to url field
        recordP += StrLen(recordP) + 1;
    }
}

```

```

recordH = DmResizeRecord(mBookmarkDB, mCurrentRecord, (nameSize + urlSize +
sizeof(Boolen));
if (!recordH) ErrFatalDisplay("Problem resizing Bookmark Record");
}
else
{
    ErrFatalDisplay("Failed to find Bookmark record while attempting to save
update");
}

// Lock down the handle and get a pointer to the new record.
recordP = MemHandleLock(recordH);
if (nameLen > 0)
    DmWrite(recordP, 0, nameP, nameSize);
else
    DmWrite(recordP, 0, kUntitledBookmarkNameP, nameSize);

if (urlP)
    DmWrite(recordP, nameSize, urlP, urlSize);
else
    DmWrite(recordP, nameSize, &zero, 1);

// write the Boolean isCached to the record
DmWrite(recordP, nameSize + urlSize, &isCached, sizeof(Boolen));

// Unlock the new record.
MemHandleUnlock(recordH);

// Release the record to the database manager. The true value indicates that
// the record contains "dirty" data. Release Record will set the record's
// dirty flag and update the database modification count.
DmReleaseRecord(mBookmarkDB, index, true);
}

/*****
* FUNCTION: deleteCurrentRecord
* DESCRIPTION: This function delete the currently selected record.
* PARAMETERS: None.
* RETURNED: nothing.
*****/
void BLBookmarkDB::deleteCurrentRecord (void)
{
    // remove current record from the database.
    DmRemoveRecord(mBookmarkDB, mCurrentRecord);
    // this causes nothing to be selected in the list.
    mCurrentRecord = kNoRecordSelected;

    // if we delete the last record, then we must set the current
    // record to be one less. Otherwise we rely on the database shifting its indexes.
    if((mCurrentRecord + 1) == numRecords) {
        mCurrentRecord--;
    }
}

/*****
* FUNCTION: bookmarkLoadRecord
* DESCRIPTION: This routine loads a bookmark entry into the appropriate
*               fields.
* PARAMETERS: frm - pointer to the Note View form
*****/

```

```

urlp = new Char(strlen(recordP) + 1);
// Copy the data from the record to the new memory chunk.
strcpy(urlp, recordP);

// Unlock the database record.
MemHandleUnlock(recordH);

// Release the record to the database system.
// (The zero means that the record data hasn't been changed.)
DbReleaseRecord(mBookmarkDB, index, 0);

return;
}

urlp = NULL;

/*****
* FUNCTION:      constructPopUpListArray
* DESCRIPTION:   This function iterates over the database and pulls out
                 the bookmark names. It puts them into the mChoicesHandle array
* PARAMETERS:   UInt index,
                 UInt numRecords,
                 UInt lstWidth,
                 mChoicesHandle
                 choicesP
* RETURNED:     Nothing.
*****/
void BLBookmarkDB::constructPopUpListArray (UInt itemIndex, UInt numRecords,
                                           UInt lstWidth, VoidHandle mChoicesHandle, CharPtr choicesP)
{
    Handle recordH;
    CharPtr recordP;
    Int i, textLen;
    Boolean fits;
    Err error;

    // A sequence of strings packed one after another, one for each record.
    for (itemIndex = 0; itemIndex <= numRecords; itemIndex++)
    {
        // the last element is the Edit Bookmark command.
        if (itemIndex == numRecords)
        {
            recordP = kEditBookmarksCommand;
        }
        else
        {
            recordH = (Handle) DbGetRecord(mBookmarkDB, itemIndex);
            recordP = (CharPtr) MemHandleLock(recordH);

            // Determine the length of text that will fit within the list bounds.
            i = (Int) lstWidth;
            textLen = (Int) strlen(recordP);
            FntCharsInWidth(recordP, i, &textLen, &fits);

            // Grow the choices buffer to accomodate the new string. We must unlock
            // the chunk so that the Memory Manager can move the chunk if necessary to
            // grow it.
            MemHandleUnlock(mChoicesHandle);
            error = MemHandleResize(mChoicesHandle, textLen + choicesOffset + sizeof('\0'));
            choicesP = (CharPtr) MemHandleLock(mChoicesHandle);

            // Check for fatal error.
            ErrFatalDisplayIf(error, "Could not grow choices for popup list.");

            // Copy the text from the record to the choices buffer.
            for (i = 0; i < textLen; i++)

```

```

        choicesP[choicesOffset + i] = recordP[i];

        // Update the end of choices offset and set a zero terminator
        // on the string in the choices buffer.
        choicesOffset += textLen;
        choicesP[choicesOffset] = 0;

        if (itemIndex != numRecords)
        {
            // Unlock the handle to the record.
            MemHandleUnlock(recordH);

            // Release the record, not dirty.
            DbReleaseRecord(mBookmarkDB, itemIndex, false);
        }
    }

/*****
* FUNCTION:      editFormListDrawItem
* DESCRIPTION:   Draw an item in the edit form's list. This routine is
                 called from LstDrawList once for each item in the list.
* PARAMETERS:   itemNum - which list item to draw
                 bounds - bounds in which to draw the item
                 itemText - pointer to data (not used)
* RETURNED:     Nothing.
*****/
void BLBookmarkDB::editFormListDrawItem (UInt itemNum, RectanglePtr bounds, CharPtr
                                           itemText)
{
    Handle rectHandle;
    CharPtr rectTextP;
    UInt x;
    Int textLen, lstWidth;
    Boolean fits;

    // Retrieve the record from the database and lock it down.
    rectHandle = (Handle) DbGetRecord(mBookmarkDB, itemNum);
    rectTextP = (CharPtr) MemHandleLock(rectHandle);

    // Determine the length of text that will fit within the list bounds.
    lstWidth = bounds->extent.x - 2;
    textLen = (Short) strlen(rectTextP);
    FntCharsInWidth(rectTextP, &lstWidth, &textLen, &fits);

    // Now draw the text from the record.
    x = (UInt) bounds->topLeft.x;
    WinDrawChars(rectTextP, (Word) textLen, (SWord) x, bounds->topLeft.y);

    // Unlock the handle to the record.
    MemHandleUnlock(rectHandle);

    // Release the record, not dirty.
    DbReleaseRecord(mBookmarkDB, itemNum, false);
}

/*****
* FUNCTION:      getFirstUrlMatch
* DESCRIPTION:   Retrieves the first element in the database that matches
                 the given string. called from LstDrawList once for each item in the
                 list.
* PARAMETERS:   urlp - url to match
* RETURNED:     CharPtr to url string or null if none.
*****/

```

```

CharPtr BLBookmarkDB::getFirstUrlMatch(CharPtr urlP)
{
    Handle recordH;
    CharPtr recordP;
    Int i, textLen;
    CharPtr matchP = NULL;
    Boolean found = false;

    // Iterate backwards to effectively search from top dom (as displayed)
    for (i = (Int)(getNumRecords() - 1); i >= 0; i--)
    {
        recordH = (Handle) DmGetRecord(mBookmarkDB, (UInt) i);
        recordP = (CharPtr) MemHandleLock(recordH);

        textLen = (Short) StrLen(recordP);
        // advance to the url
        recordP += (textLen + 1);

        if (recordP)
        {
            if (StrCompare(recordP, urlP, StrLen(urlP)) == 0)
            {
                found = true;
                matchP = BLUtils::cloneString(recordP);
            }
        }

        // Unlock the handle to the record.
        MemHandleUnlock(recordH);

        // Release the record, not dirty.
        DmReleaseRecord(mBookmarkDB, (UShort) i, false);

        if (found) break;
    }

    if (matchP) return matchP;
    return NULL;
}

#pragma once
#include "BLUtils.h"

static DmOpenRef mBookmarkDB; // handle for database

class BLBookmarkDB
{
public:
    BLBookmarkDB();
    ~BLBookmarkDB();

    UInt
    void
    getCurrentRecord(void);
    setCurrentRecord(UInt index);

    Boolean
    UInt
    isValidRecord(void);
    getNumRecords(void);

    void
    openBookmarkDB(void);
    void
    closeBookmarkDB(void);
    void
    saveBookmarkRecord(Word nameFieldID, Word urlFieldID, Boolean
    isNewRecord, Boolean isCached);
    loadBookmarkRecord(Word nameField, Word urlField, Word checkboxID);
    getUriForRecord(Word index, CharPtr& urlP);
    deleteCurrentRecord(void);

    constructPopUpListArray (UInt itemIndex, UInt numRecords, UInt
    lstWidth,
        voidHandle mChoicesHandle, CharPtr& choicesP);

    // Drawing callback functions
    static void
    editFormListDrawItem (UInt itemNum, RectanglePtr bounds, CharPtr
    *itemText);
    static void
    popUpListDrawItem (UInt itemNum, RectanglePtr bounds, CharPtr
    *itemText);

    // auto complete
    CharPtr
    getFirstUrlMatch(CharPtr urlP);
}

```



```

}

void free(void * ptr)
{
    MemPtrFree(ptr);
} // BLCacheDB.cpp

#include "BLCacheDB.h"
#include "BrowserApp.h"
BLCacheDB::BLCacheDB()
{
    mCurrentRecord = -1;
}

BLCacheDB::~BLCacheDB()
{
    DbDeleteDatabase(0, mDBID);
}

void
BLCacheDB::open()
{
    BLDATABASE::open(BrowserApp::appFileCreator, 'cach', "BLCache", dbModeReadWrite, true);
}

CharPtr
BLCacheDB::newRecord( BLVector<BLHistoryItem *, kHistorySize> & pageHistory,
                      int currPage)
{
    OpenCacheRecord * cacheRecordP = new OpenCacheRecord();

    if(cacheRecordP == NULL)
        throw OutOfMemoryException();

    cacheRecordP->mHandle = NULL;
    cacheRecordP->mPhysicalSize = 64000;
    cacheRecordP->mLogicalSize = 0;
    cacheRecordP->mIndex = (Word) -1;
    cacheRecordP->mRelease = true;
    cacheRecordP->mDelete = true;

    VoidHand openRecordHandle;
    Word index;

    for(int i=0; i<2; i++)
    {
        index = dmMaxRecordIndex;
        openRecordHandle = DmNewRecord(mRef, &index, cacheRecordP->mPhysicalSize);

        if(openRecordHandle != NULL)
        {
            return addToCache(cacheRecordP, openRecordHandle, index);
        }

        cleanCache(pageHistory, currPage, i);
    }

    while(true)
    {
        index = dmMaxRecordIndex;
        openRecordHandle = DmNewRecord(mRef, &index, cacheRecordP->mPhysicalSize);

        if(openRecordHandle != NULL)
        {
            return addToCache(cacheRecordP, openRecordHandle, index);
        }
    }

    cacheRecordP->mPhysicalSize /= 2;
    if(cacheRecordP->mPhysicalSize == 0)
        break;
}

```

```

        cacheRecordP->mPhysicalSize = 0;
        delete cacheRecordP;
        return NULL;
    }

    CharPtr
    BLCacheDB::addToCache(OpenCacheRecord * recordP, VoidHand handle, Word index)
    {
        recordP->mHandle = handle;
        recordP->mIndex = index;
        mCurrentRecord = mOpenRecords.getNumElements();
        mOpenRecords.add(recordP);
        recordP->mData = (CharPtr) MemHandleLock(handle);
        return recordP->mData;
    }

    CharPtr
    BLCacheDB::checkInCurrent(bool canMove)
    {
        if(mCurrentRecord == -1)
            return NULL;

        OpenCacheRecord * recordP = mOpenRecords[mCurrentRecord];
        recordP->mDelete = false;
        if(canMove)
        {
            MemHandleUnlock(recordP->mHandle);
            recordP->mHandle = DmResizeRecord(mRef, recordP->mIndex, recordP->mLogicalSize);
            recordP->mPhysicalSize = recordP->mLogicalSize;
            mTotalCacheSize += recordP->mLogicalSize;
            recordP->mData = (CharPtr) MemHandleLock(recordP->mHandle);
            return recordP->mData;
        }
        return NULL;
    }

    void
    BLCacheDB::removeCurrent()
    {
        if(mCurrentRecord == -1)
            return;

        OpenCacheRecord * recordP = mOpenRecords[mCurrentRecord];
        MemHandleUnlock(recordP->mHandle);
        if(recordP->mRelease)
            DmReleaseRecord(mRef, recordP->mIndex, true);
        decrementRecords(recordP->mIndex);
        DmRemoveRecord(mRef, recordP->mIndex);
        delete mOpenRecords.remove(mCurrentRecord);
        mCurrentRecord = -1;
    }

    bool
    BLCacheDB::growRecord(UInt newSize)
    {

```

```

        if(mCurrentRecord == -1)
            return false;
        mOpenRecords[mCurrentRecord]->mLogicalSize = newSize;
        if(newSize > mOpenRecords[mCurrentRecord]->mPhysicalSize)
            return false;
        else
            return true;
    }

    void
    BLCacheDB::decrementRecords(UInt index)
    {
        BLIterator<OpenCacheRecord *, Kincrement> iter(mOpenRecords);
        while(iter.hasNext())
        {
            OpenCacheRecord * recordP = iter.getNext();
            if(recordP->mIndex > index)
                recordP->mIndex--;
        }
    }

    OpenCacheRecord *
    BLCacheDB::findRecordOpen(CharPtr url, Word index)
    {
        BLIterator<OpenCacheRecord *, Kincrement> iter(mOpenRecords);
        while(iter.hasNext())
        {
            OpenCacheRecord * recordP = iter.getNext();
            if(url && recordP->mData)
            {
                if(StrCompare(url, recordP->mData) == 0)
                    return recordP;
            }
            else if(recordP->mIndex == index)
                return recordP;
        }
        return NULL;
    }

    void
    BLCacheDB::closeRecords()
    {
        while(mOpenRecords.getNumElements() > 0)
        {
            OpenCacheRecord * recordP = mOpenRecords[0];
            MemHandleUnlock(recordP->mHandle);
            if(recordP->mRelease)
            {
                if(!recordP->mDelete)
                {
                    DmResizeRecord(mRef, recordP->mIndex, recordP->mLogicalSize);
                    mTotalCacheSize += recordP->mLogicalSize;
                }
                DmReleaseRecord(mRef, recordP->mIndex, true);
            }
            if(recordP->mDelete)
            {
                decrementRecords(recordP->mIndex);
                DmRemoveRecord(mRef, recordP->mIndex);
            }
        }
    }

```

```

        delete mOpenRecords.remove(0);
        mCurrentRecord = -1;
        mOpenRecords.deleteAll();
    }

    UInt
    BLCacheDB::findRecord(CharPtr url, CharPtr &cacheEntry)
    {
        UInt numRecords = DmNumRecords(mRef);
        UInt dbIndex = 0;
        // first see if it's already open, then look it cache.
        OpenCacheRecord * cacheRecordP = isRecordOpen(url, 0);
        if(!cacheRecordP != NULL) && (cacheRecordP->mPhysicalSize > 0)
        {
            cacheEntry = cacheRecordP->mData;
            return cacheRecordP->mPhysicalSize;
        }
        for(USHort dbIndex=0; dbIndex < numRecords; dbIndex++)
        {
            Handle charHandle = (Handle) DmQueryRecord(mRef, dbIndex);
            if(charHandle == NULL)
                continue;
            CharPtr string = (CharPtr) MemHandleLock(charHandle);
            if(StrCompare(string, url) == 0)
            {
                cacheRecordP = new OpenCacheRecord();
                if(cacheRecordP == NULL)
                    throw OutOfMemoryException();
                cacheRecordP->mHandle = charHandle;
                cacheRecordP->mLogicalSize = cacheRecordP->mPhysicalSize = MemPtrSize(string);
                cacheRecordP->mIndex = dbIndex;
                cacheRecordP->mRelease = false;
                cacheRecordP->mDelete = false;
                cacheRecordP->mData = string;
                cacheEntry = string;
                mOpenRecords.add(cacheRecordP);
                return cacheRecordP->mPhysicalSize;
            }
            MemHandleUnlock(charHandle);
            cacheEntry = NULL;
            return 0;
        }

        void
        BLCacheDB::cleanCache( BLVector<BLHistoryItem *, KHistorySize> &pageHistory,
                             int curPage,
                             int attempt)
        {
            UInt numRecords = DmNumRecords(mRef);
            UInt dbIndex = 0;
            for(USHort dbIndex=0; dbIndex < numRecords; dbIndex++)
            {

```

```

        if(!isRecordOpen(NULL, dbIndex) != NULL)
            continue;

        Handle charHandle = (Handle) DmQueryRecord(mRef, dbIndex);

        if(charHandle == NULL)
            continue;

        CharPtr string = (CharPtr) MemHandleLock(charHandle);

        BLIterator<BLHistoryItem *, kHistorySize> iter(pageHistory);

        bool keep = false;
        if(attempt == 0)
        {
            int pos = 0;

            while(iter.hasNext())
            {
                BLHistoryItem * item = iter.getNext();

                if(StrCompare(item->getUrl(), string) == 0)
                {
                    if(pos < currPage)
                    {
                        keep = true;
                        break;
                    }
                }
                pos++;
            }
        }

        MemHandleUnlock(charHandle);

        if(!keep)
        {
            decrementRecords(dbIndex);
            DmRemoveRecord(mRef, dbIndex);
            dbIndex--;
        }
    }
}

```

```
// BLCacheDB.h
```

```
#pragma once
```

```
#include "BLDatabase.h"
#include "BLVector.h"
#include "WMLState.h"

```

```
class OpenCacheRecord
```

```
{
public:
```

```
    VoidHand mHandle;
    CharPtr mData;
    ULONG mPhysicalSize;
    ULONG mLogicalSize;
    Word mIndex;
    bool mRelease;
    bool mDelete;
};

```

```
class BLCacheDB : public BLDatabase
```

```
{
public:
```

```
    enum { kHistorySize = 20, kIncrement = 4 };

```

```

        BLCacheDB();
        ~BLCacheDB();

        virtual void
        open();

        CharPtr newRecord( BLVector<BLHistoryItem *, kHistorySize> & pageHistory,
            int currPage);

        bool growRecord(ULONG newSize);
        void closeRecords();

        ULONG findRecord(CharPtr url, CharPtr &cacheEntry);

        Word bytesLeft() { return
            (Word)mOpenRecords[mCurrentRecord]->mPhysicalSize -
            (Word)mOpenRecords[mCurrentRecord]->mLogicalSize; }

        CharPtr addToCache(OpenCacheRecord * recordP, VoidHand handle, Word index);

        CharPtr checkInCurrent(bool canMove = false);
        void removeCurrent();

        protected:

        OpenCacheRecord * isRecordOpen(CharPtr url, Word index);

        void decrementRecords(ULONG index);

        BLVector<OpenCacheRecord *, kIncrement> mOpenRecords;

        void cleanCache( BLVector<BLHistoryItem *, kHistorySize> & pageHistory,
            int currPage,
            int attempt);

        ULONG mTotalCacheSize;
        int mCurrentRecord;
    };
}
// BLDatabase.cpp
#include "BLDatabase.h"

```

```
void
BLDatabase::open(
    ULONG appType,
    ULONG dbType,
    CharPtr dbName,
    UInt mode,
    bool deleteExisting)
{
    Err error;
    UInt cardNo;
    UInt dbAttrs;

    // Find the bookmark database.
    mRef = DmOpenDatabaseByTypeCreator(dbType, appType, mode);

    if(deleteExisting && mRef)
    {
        error = DmOpenDatabaseInfo(mRef, &dbID, NULL, &cardNo, NULL);

        DmCloseDatabase(mRef);
        DmDeleteDatabase(0, mDBID);

        mRef = NULL;
    }

    if(!mRef)
    {
        // The database doesn't exist, create it now.
        error = DmCreateDatabase(0, dbName, appType, dbType, false);

        // Check for fatal error.
        ErrFatalDisplayIf(error, "Could not create new docs database.");
    }
}

```

```
// Find the application's database.
mRef = DmOpenDatabaseByTypeCreator(dBType, appType, mode);

// Get info about the database
error = DmOpenDatabaseInfo(mRef, &mDBID, NULL, NULL, &cardNo, NULL);

// Check for fatal error.
ErrFatalDisplayIf(error, "Could not get database info.");

// Get attributes for the database
error = DmDatabaseInfo(0, mDBID, NULL, &dbAttrs, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

// Check for fatal error.
ErrFatalDisplayIf(error, "Could not get database info.");
}
```

```
void
BLDatabase::close()
{
    DmCloseDatabase(mRef);
}
```

```
.Database.h
```

```
#pragma once
```

```
class BLDatabase
```

```
{
public:
```

```
    virtual ~BLDatabase() {}
```

```
    virtual void open() = 0;
```

```
    virtual void close();
```

```
protected:
```

```
    void open(
        ULONG appType,
        ULONG dBType,
        CharPtr dbName,
        UInt mode = dmModeReadWrite,
        bool deleteExisting = false);
```

```
    DmOpenRef mRef;
    LocalID mDBID;
```

```
}; // BLEventHandler.h
```

```
ma once
```

```
class BLEventHandler
```

```
{
public:
```

```
    enum EventClasses { kFormType, kMenuItem, kUIType };
```

```
    virtual bool doesHandleType(EventClasses /*event*/) { return false; }
```

```
    virtual bool doesHandleObject(Word /*objectId*/) { return false; }
```

```
    virtual Boolean handleEvent(EventPtr eventP) = 0;
```

```
}; // BLForm.cpp
```

```
#include "BLForm.h"
```

```
#include "BLIterator.h"
```

```
#include "BLUtils.h"
```

```
BLForm::BLForm(UInt supportedFormID)
```

```
{
    mSupportedFormID = supportedFormID;
    mFocus = NULL;
}
```

```
BLForm::BLForm()
{
}
```

```
void
BLForm::formLoad(FormPtr formP)
{
    mFormPtr = formP;
}
```

```
void
BLForm::formOpen()
{
    draw();
}
```

```
void
BLForm::formClose()
{
}
```

```
Boolean
BLForm::handleEvent(EventPtr eventP)
{
    Boolean handled = false;

    if(mFocus != NULL)
    {
        handled = mFocus->handleEvent(eventP);
    }

    return handled;
}
```

```
void
BLForm::draw()
{
    BLIterator<BLView *, kViewsInc> iter(mViews);

    while(iter.hasNext())
    {
        iter.getNext()->draw();
    }
}
```

```
void
BLForm::addView(BLView * view)
{
    mViews.add(view);
}
```

```
VoidPtr
BLForm::getObjectPtr(Word objectId)
{
    return FrmGetObjectPtr(mFormPtr, FrmGetObjectIndex(mFormPtr, objectId));
}
```

```
void
BLForm::setTitle(CharPtr newTitle)
{
    RectangleType originalRect;
    WinGetClip(&originalRect);
    WinResetClip();
}
```

```

    CharPtr oldTitle = FrmGetTitle(FrmGetActiveForm());
    FrmSetTitle(FrmGetActiveForm(), BLUtils::cloneString(newTitle));
    delete[] oldTitle;

    WinSetClip(&originalRect);
}

```

```

void
BLForm::refreshFormPointer()
{
    mFormPtr = FrmGetActiveForm();
} // BLForm.h

#pragma once

#include "BLEventHandler.h"
#include "BLView.h"
#include "BLVector.h"

```

```

class BLForm : public BLEventHandler
{
public:

```

```

    enum { kViewsInc = 4 };

    virtual
        BLForm(UINT supportedFormID);
        ~BLForm();

    virtual void
        formLoad(FormPtr formP);
    virtual void
        formOpen();
    virtual void
        formClose();

    virtual bool
        getObjectPtr(Word objectID);
    virtual bool
        getFormPtr() { return mFormPtr; }

    virtual bool
        doesHandleType(EventClasses event)
        { return event == kFormType; }

    virtual bool
        doesHandleObject(Word objectID)
        { return objectID == mSupportedFormID; }

    virtual Boolean
        handleEvent(EventPtr eventP);

    virtual void
        draw();
    virtual void
        addView(BLView * view);

    static void
        setTitle(CharPtr newTitle);

    void
        refreshFormPointer();
}

```

rotected:

```

    BLVector<BLView*, kViewsInc> mViews;
    BLView* mFocus;
    FormPtr mFormPtr;
    Word mSupportedFormID;
}; // BLGlyph.cpp

#include "BLGlyph.h"
#include "BLUtils.h"
#include "BLWMLSession.h"
#include "BLWMLTable.h"
#include "BLWMLActions.h"
#include "BLWMLInput.h"

const int kItalicInterval = 2;

BLMemoryManager
BLGlyph::sMemManager;

BLWMLSession *

```

```

BLGlyph::sSession = NULL;
WMLAction * sClickedAction = NULL;
BLGlyph::sClickedAction = NULL;
BLVector<RectangleType, 4>
BLGlyph::sActiveRectVector(0);

BLGlyph::HitLiteAction
BLGlyph::sHitLiteAction = kFindSelection;

BLGlyph::BLGlyph(BLGlyph * parent)
{
    if (parent && parent->children())
        parent->children()->add(this);
}

void
BLGlyph::setMemoryManagers()
{
    // !!! Make sure you got'em all !!!
    BLVector<BLGlyph*, kChildrenInc>::setMemManager(&sMemManager);
    BLVector<WMLDo*, kChildrenInc>::setMemManager(&sMemManager);
    BLVector<WMLImage*, kChildrenInc>::setMemManager(&sMemManager);
    BLVector<TextState, kChildrenInc>::setMemManager(&sMemManager);
}

```

```

void *
BLGlyph::operator new(size_t size)
{
    return sMemManager.newChunk(size);
}

```

```

void
BLGlyph::operator delete(void *, size_t)
{
}

```

```

RectangleType
BLGlyph::getBounds(const Point topLeft)
{
    RectangleType rect;
    rect.topLeft = topLeft;
    rect.extent = getExtent();
    return rect;
}

```

```

bool
BLGlyph::intersects(Point topLeft, PointType p)
{
    const RectangleType & bounds = getBounds(topLeft);
    Point pt(p);
    return pt.isInside(bounds);
}

```

```

Point
BLGlyph::getExtent(const Point &areaExtent)
{
    if (children() == NULL)
        return Point::sZeroPoint();
    BLIterator<BLGlyph*, kChildrenInc> iter('children());
    Point totalExtent;
}

```

```

while(iter.hasNext())
{
    BLGlyph * element = iter.getNext();
    Point extent = element->getExtent(areaExtent);
    totalExtent.y += extent.y;
    if(extent.x > totalExtent.x)
        totalExtent.x = extent.x;
}
return totalExtent;
}

```

```

bool
BLGlyph::drawOrClick( const RectangleType & displayArea,
    const Point &offset,
    PointType P,
    bool isDraw)
{
    if(children() == NULL)
        return false;
    BLIterator<BLGlyph*, kChildrenInc> iter(*children());
    RectangleType areaLeft = displayArea;
    Point remainingOffset = offset;

```

```

while(iter.hasNext())
{
    BLGlyph * child = iter.getNext();
    int height = child->getExtent().y;
    if(remainingOffset.y > height)
    {
        remainingOffset.y -= height;
        continue;
    }

```

```

    if(isDraw)
        child->draw(areaLeft, remainingOffset);
    else
    {
        if(child->click(areaLeft, remainingOffset, p))
            return true;
    }
    Point extent = child->getExtent(displayArea.extent);
    areaLeft.topLeft.y += extent.y - remainingOffset.y;
    areaLeft.extent.y -= extent.y - remainingOffset.y;
    remainingOffset.y = 0;
    if(areaLeft.extent.y <= 0)
        break;
    return false;
}

```

```

void
BLGlyph::draw(const RectangleType & displayArea, const Point &offset)
{
    drawOrClick(displayArea, offset, Point::zeroPoint(), true);
}

```

```

bool
BLGlyph::click(const RectangleType & displayArea,

```

```

const Point &offset,
    PointType p)
{
    return drawOrClick(displayArea, offset, p, false);
}

```

```

CharPtr
BLGlyph::cloneString(CharPtr str)
{
    if(str == NULL)
        return NULL;

```

```

    CharPtr copy = (CharPtr) sMemManager.newChunk(sizeof(Char) * (StrLen(str) + 1));
    return StrCopy(copy, str);
}

```

```

void
BLGlyph::flushAll()
{
    WMLLine::sUnusedLines.deleteAll();
    WMLTextRun::sUnusedRuns.deleteAll();
    sMemManager.flushAll();
}

```

```

RectangleType
BLGlyph::getArea(const RectangleType & displayArea, const Point &offset)
{
    PointType topLeft = displayArea.topLeft;
    RectangleType rect = getBounds(topLeft);
    RectUtils::shrink(rect, offset);
    if(displayArea.extent.x < rect.extent.x)
        rect.extent.x = displayArea.extent.x;
    if(displayArea.extent.y < rect.extent.y)
        rect.extent.y = displayArea.extent.y;
    return rect;
}

```

```

WMLRoot::WMLRoot()
: BLParentGlyph(NULL),
  mNamedActions(0)
{
    mActiveCard = NULL;
    mDefaultCard = NULL;
    mTemplate = NULL;
    mDomainAccess = NULL;
    mPathAccess = NULL;
}

```

```

void
WMLRoot::draw(const RectangleType & displayArea, const Point &offset)
{
    if(mDefaultCard == NULL)
    {
        BLIterator<BLGlyph*, kChildrenInc> iter(*children());
        while(iter.hasNext())
        {
            BLGlyph * child = iter.getNext();
            WMLCard * card = dynamic_cast<WMLCard *>(child);
            if(card != NULL)
            {
                mDefaultCard = card;
            }

```

```

        mDefaultCard->activate();
        break;
    }
}

if(mActiveCard == NULL)
    mActiveCard = mDefaultCard;

if(mActiveCard)
    mActiveCard->draw(displayArea, offset);

TextState::enableBasicState();

bool
WMLRoot::click(const RectangleType & displayArea,
               const Point & offset,
               PointType p)
{
    if(mActiveCard)
        return mActiveCard->click(displayArea, offset, p);
    return false;
}

Point
WMLRoot::getExtent(const Point &areaExtent)
{
    if(mActiveCard)
        return mActiveCard->getExtent(areaExtent);
    else
    {
        Point zero(0, 0);
        return zero;
    }
}

TextState::enableBasicState();

void
WMLRoot::addAction(WMLDo * doP)
{
    mNamedActions.add(doP);
}

WMLCard::WMLCard(BLGlyph * parent, const WMLCard & original, WMLRoot * root)
: BLParentGlyph(parent),
  mNamedActions(root->getNamedActions()),
  mEvents()
{
    mNewContext = original.mNewContext;
    mID = original.mID;
    mTitle = original.mTitle;
    mTitleSet = original.mTitleSet;
    mIsOrdered = original.mIsOrdered;
    mTimer = NULL;
}

void
WMLCard::addAction(WMLDo * doP)
{
    BLIterator<WMLDo*, kChildrenInc> iter(mNamedActions);
    int i=0;
    // So we've inherited zero or more actions from the template
    // defined at the top of the deck. If this added action has the
    // same name as one those actions, replace the original.
    // Otherwise, append.
    // !!! replace the whole thing with a hashtable someday, maybe...
    if(dof->mName != NULL)
    {
        while(iter.hasNext())
        {
            WMLDo* wmldo = iter.getNext();
            if(wmldo->mName != NULL)
            {

```

```

        if(StrCompare(wmldo->mName, doP->mName) == 0)
        {
            mNamedActions.Replace(doP, i);
            return;
        }
        ++i;
    }
}

void WMLCard::draw(const RectangleType & displayArea, const Point &offset)
{
    if(!mIsTitleSet)
    {
        CharPtr oldTitle = FrmGetTitle(FrmGetActiveForm());
        CharPtr newTitle = mTitle;
        if(newTitle == NULL)
            newTitle = BLUtils::cloneString("Blazer");
        else
        {
            FntSetFont(boldFont);
            WordLength = FntWordWrap(newTitle, 78);
            FntSetFont(stdFont);

            if(length == StrLen(newTitle))
                newTitle = BLUtils::cloneString(newTitle);
            else
            {
                CharPtr tempTitle = new Char[length + 4];

                if(tempTitle == NULL)
                    throw OutOfMemoryException();

                StrNCopy(tempTitle, newTitle, length);
                newTitle = tempTitle;
                newTitle[length++] = '.';
                newTitle[length++] = '.';
                newTitle[length++] = '.';
                newTitle[length] = '\0';
            }
        }

        RectangleType originalRect;
        WinGetClip(&originalRect);
        WinResetClip();

        FrmSetTitle(FrmGetActiveForm(), newTitle);

        WinSetClip(&originalRect);

        delete[] oldTitle;
        mIsTitleSet = true;
    }

    BLGlyph::draw(displayArea, offset);
}

void WMLCard::activate()
{
    if(mTimer)
        mTimer->start();
}

void WMLCard::deactivate()
{

```

```

        if(mTimer)
            mTimer->stop();
        mIsTitleSet = false;
    }

    void WMLCard::doneParsing()
    {
        BLIterator<WMLDo*, kChildrenInc> iter(mNamedActions);

        if(!iter.hasNext())
            return;

        WMLParagraph * doParagraph = new WMLParagraph(this);

        if(doParagraph == NULL)
            throw OutOfMemoryException();

        TextState state;
        state.underline = grayUnderline;

        if(new WMLBreak(doParagraph) == NULL)
            throw OutOfMemoryException();

        bool hasButton = false;
        while(iter.hasNext())
        {
            WMLDo * doElem = iter.getNext();

            if(doElem->getAction() != NULL)
            {
                hasButton = true;

                state.action = doElem;
                CharPtr itemName = doElem->mLabel;

                if(itemName == NULL)
                    itemName = "Unknown Action";

                new WMLButton( doParagraph,
                               itemName,
                               (int) StrLen(itemName),
                               state);
            }
        }

        if(hasButton)
            new WMLBreak(doParagraph);
    }

    void WMLCard::onEnterEvent(NavigationAction lastAction)
    {
        switch(lastAction)
        {
            case kGoBack:
                if(mEvents.mOnEnterBackward)
                    mEvents.mOnEnterBackward->activate();
                break;
            default:
                if(mEvents.mOnEnterForward)
                    mEvents.mOnEnterForward->activate();
                break;
        }
    }
}

```



```

WMLParagraph::WMLParagraph(BLGlyph * parent)
: BIParentGlyph(parent),
mLines()
{
    bits.mFlowDone = false;
    bits.mWrapOn = true;
    mAlign = kLeft;

    bits.mIsExtentCached = false;
    mTotalExtent.x = mTotalExtent.y = 0;

    mFlowVars = NULL;
}

WMLParagraph::WMLParagraph(BLGlyph * parent, const WMLParagraph &original)
: BIParentGlyph(parent)
{
    bits.mFlowDone = false;
    bits.mIsExtentCached = false;
    mFlowVars = NULL;

    bits.mIsWrapOn = original.bits.mIsWrapOn;
    mAlign = original.mAlign;

void
WMLParagraph::draw(const RectangleType & displayArea, const Point &offset)
{
    drawOrClick(displayArea, offset, Point::ZeroPoint(), true);
}

bool
WMLParagraph::click(const RectangleType & displayArea,
                    const Point &offset,
                    PointType p)
{
    return drawOrClick(displayArea, offset, p, false);
}

void
WMLParagraph::needsReflow()
{
    bits.mFlowDone = false;
    bits.mIsExtentCached = false;

    // salvage the memory:
    WMLParagraph::WMLParagraph * kChildrenInc = new WMLParagraph * (mLines.children());
    while(iter.hasNext())
    {
        WMLLine * line = dynamic_cast<WMLLine *>(iter.getNext());
        if(line != NULL)
        {
            BLIterator<BLGlyph*, kChildrenInc> lineIter(*line->children());
            while(lineIter.hasNext())
            {
                BLGlyph * glyph = lineIter.getNext();
                if(dynamic_cast<WMLTextRun *>(glyph) != NULL)
                {
                    WMLTextRun::sUnusedRuns.add((WMLTextRun *) glyph);
                }
            }
            WMLLine::sUnusedLines.add(line);
        }
    }
    mLines.children()->removeAll();
}

```

```

WMLParagraph::drawOrClick(const RectangleType & displayArea,
                        const Point &offset,
                        PointType p,
                        bool isDraw)
{
    PointType areaExtent = displayArea.extent;
    flow(areaExtent);

    BLIterator<BLGlyph*, kChildrenInc> iter(*mLines.children());
    RectangleType areaLeft = displayArea;
    Point remainingOffset = offset;
    while(iter.hasNext())
    {
        WMLLine * element = dynamic_cast<WMLLine *>(iter.getNext());
        LineMetrics metrics = element->getMetrics();
        int height = metrics.height();
        if(remainingOffset.y > height)
        {
            remainingOffset.y -= height;
            continue;
        }
        if(!isDraw)
            element->draw(areaLeft, remainingOffset);
        else
        {
            if(element->click(areaLeft, remainingOffset, p))
                return true;
        }
        if(remainingOffset.y > 0)
        {
            areaLeft.topLeft.y += height - remainingOffset.y;
            areaLeft.extent.y -= height - remainingOffset.y;
            remainingOffset.y -= height;
        }
        if(remainingOffset.y < 0)
            remainingOffset.y = 0;
        else
        {
            areaLeft.topLeft.y += height;
            areaLeft.extent.y -= height;
        }
        if(areaLeft.extent.y <= 0)
            break;
        flow(areaExtent);
        return false;
    }

    Point
    WMLParagraph::getExtent()
    {
        BLIterator<BLGlyph*, kChildrenInc> iter(*mLines.children());
        if(bits.mIsExtentCached)
            return mTotalExtent;
        Point totalExtent;
        while(iter.hasNext())
        {

```

```

WMLLine * element = dynamic_cast<WMLLine *>(iter.getNext());
Point extent = element->getExtent();
totalExtent.y += extent.y;
if(extent.x > totalExtent.x)
    totalExtent.x = extent.x;
}
if(bits.mFlowDone)
{
    totalExtent.y += 5;
    mTotalExtent = totalExtent;
    bits.misExtentCached = true;
}
return totalExtent;
}
agraph::getExtent(const Point &areaExtent)
{
    if(areaExtent != Point::zeroPoint())
        flow(areaExtent);
    return getExtent();
}
void
WMLParagraph::newLine(bool doComputeMetrics)
{
    if(doComputeMetrics)
        mFlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
    mFlowVars->mCurrentTextState.enable();
    mFlowVars->mCurrentLine = new WMLLine(&mLines, mAlign);
    mFlowVars->mWidthLeft = mFlowVars->mDefaultWidth;
}
bool
WMLParagraph::flow(const Point &areaExtent)
{
    if(bits.mFlowDone)
        return true;
    if(this == WBXMLParser::sGetCurrentNode())
        return false;
    // Mimics a series of recursive calls with a stack of iterators.
    // Walks tree in order.
    BLIteratorNode<BLGlyph*, kChildrenInc> * curIter;
    BLIteratorNode<BLGlyph*, kChildrenInc> * iter;
    if(mFlowVars == NULL)
    {
        mFlowVars = new ParagraphFlowVars();
        mFlowVars->mCurrentTextState.bits.wordWrap = bits.misWrapOn;
        mFlowVars->mWidthLeft = mFlowVars->mDefaultWidth = areaExtent.x;
        iter = new BLIteratorNode<BLGlyph*, kChildrenInc> (mChildren);
        mFlowVars->mIterStack = new BLStack;
        mFlowVars->mIterStack->push(iter);
        mFlowVars->mCurrentLine = new WMLLine(&mLines, mAlign);
        bits.mUnwrappedSpillover = false;
    }
    else
    {

```

```

if(mFlowVars->mCurrentLine == NULL)
    mFlowVars->mCurrentLine = new WMLLine(&mLines, mAlign);
mFlowVars->mCurrentTextState.enable(); // enable plain state
while(mFlowVars->mIterStack->top() != NULL)
{
    curIter = (BLIteratorNode<BLGlyph*, kChildrenInc> *) mFlowVars->mIterStack->pop();
    curIter->refresh();
    while(curIter->hasNext())
    {
        BLGlyph * elem = curIter->getNext();
        if(elem == WBXMLParser::sGetCurrentNode())
        {
            // this is still being parsed.
            curIter->backup();
            mFlowVars->mIterStack->push(curIter);
            mFlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
            return false;
        }
        if(elem->children())
        {
            if(dynamic_cast<WMLTable *>(elem) != NULL)
            {
                if(WBXMLParser::sGetNumOpenTables() > 0)
                {
                    curIter->backup();
                    mFlowVars->mIterStack->push(curIter);
                    mFlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
                    return false;
                }
                WMLTable * table = dynamic_cast<WMLTable *>(elem);
                if(mFlowVars->mCurrentLine->children()->getNumElements() > 0)
                {
                    mFlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
                    mFlowVars->mCurrentTextState.enable();
                    mFlowVars->mCurrentLine = new WMLLine(&mLines, mAlign);
                }
                mFlowVars->mCurrentLine->add(table);
                mFlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
                mFlowVars->mCurrentTextState.enable();
                Point tableExtent(mFlowVars->mDefaultWidth, 0);
                table->flow(&mLines, tableExtent);
                newLine(false);
            }
            else
            {
                if(dynamic_cast<WMLSelect *>(elem) != NULL)
                {
                    WMLSelect * selectP = dynamic_cast<WMLSelect *>(elem);
                    newLine();
                    if(selectP->getTitle())
                    {
                        WMLTextRun * textRun = new WMLTextRun( mFlowVars->mCurrentLine,
                                                                selectP->getTitle(),
                                                                selectP->getTitleLength());
                        mFlowVars->mCurrentTextState);
                        flowText(textRun);
                        newLine();
                    }
                }
            }

```

```

selectP);
    if(selectP->isDisplayable())
    {
        // the select itself will display UI, not the children:
        selectP->computeMetrics(mFlowVars->mWidthLeft);
        mFlowVars->mCurrentLine->children()->add((WMLLineElement *)
            newLine());
        continue;
    }
    else if(dynamic_cast<WMLOption *>(elem) != NULL)
    {
        WMLOption * optionP = dynamic_cast<WMLOption *>(elem);
        newLine();
        mFlowVars->mCurrentLine->children()->add(elem);
        mFlowVars->mWidthLeft -= optionP->getExtent(areaExtent).x;
        if(optionP->getTitleLength() > 0)
        {
            WMLTextRun * textRun = new WMLTextRun( mFlowVars->mCurrentLine,
                optionP->getTitle(),
                optionP->getTitleLength());
            mFlowVars->mCurrentTextState();
            flowText(textRun);
        }
        iter = new BLIteratorNode<BLGlyph*, kChildrenInc> (*elem->children());
        if(curIter->hasNext())
        {
            mFlowVars->mIterStack->push(curIter);
            curIter = NULL; // make it NULL so it won't be deleted
        }
        mFlowVars->mIterStack->push(iter);
        break;
    }
    else
    {
        // It's a leaf, so flow it.
        if(dynamic_cast<WMLTextRun *>(elem) != NULL)
        {
            flowText(dynamic_cast<WMLTextRun *>(elem));
        }
        else if(dynamic_cast<WMLBreak *>(elem) != NULL)
        {
            WMLTextRun * newRun = new WMLTextRun((BLParentGlyph
                mFlowVars->mCurrentLine, "", 0, mFlowVars->mCurrentTextState);
            newRun->computeMetrics(mFlowVars->mWidthLeft);
            newLine();
        }
        else if(dynamic_cast<WMLButton *>(elem) != NULL)
        {
            WMLButton * buttonP = dynamic_cast<WMLButton *>(elem);
            LineMetrics metrics = buttonP->getMetrics();
            if( (metrics.width >= mFlowVars->mWidthLeft) &&
                (metrics.width <= mFlowVars->mDefaultWidth))
                newLine();
            mFlowVars->mCurrentLine->children()->add((WMLLineElement *) buttonP);
            mFlowVars->mWidthLeft -= metrics.width;
        }
        else if(dynamic_cast<WMLInput *>(elem) != NULL)
    }

```

```

WMLInput * inputP = dynamic_cast<WMLInput *>(elem);
inputP->computeMetrics(mFlowVars->mWidthLeft);
mFlowVars->mCurrentLine->children()->add(elem);
newLine();
    else if(dynamic_cast<WMLFieldSet *>(elem) != NULL)
    {
        WMLFieldSet * fieldSetP = dynamic_cast<WMLFieldSet *>(elem);
        newLine();
        if(fieldSetP->getTitleLength() > 0)
        {
            WMLTextRun * textRun = new WMLTextRun( mFlowVars->mCurrentLine,
                fieldSetP->getTitle(),
                fieldSetP->getTitleLength());
            mFlowVars->mCurrentTextState();
            flowText(textRun);
        }
        newLine();
        else if(dynamic_cast<WMLImage *>(elem) != NULL)
        {
            WMLImage * image = dynamic_cast<WMLImage *>(elem);
            Point extent = image->getExtent();
            mFlowVars->mCurrentTextState.setWordWrap(bits.misWrapOn);
            if(mFlowVars->mCurrentTextState.bits.wordWrap && (extent.x >
                mFlowVars->mWidthLeft))
            {
                // Image wider than space available, so go to next line:
                newLine();
            }
            if(!image->isDownloaded())
            {
                int altTextLen = 0;
                if(image->getAltText() != NULL)
                    altTextLen = (int) StrLen(image->getAltText());
                if(altTextLen > 0)
                {
                    WMLTextRun * newRun;
                    CharPtr altText = image->getAltText();
                    if(altText[0] != '\0')
                    {
                        newRun = new WMLTextRun(NULL,
                            "\n",
                            1,
                            image->getTextState());
                        flowText(newRun);
                    }
                    newRun = new WMLTextRun(NULL,
                        altText,
                        altTextLen,
                        image->getTextState());
                    flowText(newRun);
                }
                if(altText[altTextLen - 1] != '\0')
                {
                    newRun = new WMLTextRun(NULL,
                        "\n",
                        1,
                        image->getTextState());
                    flowText(newRun);
                }
            }
        }
    }

```

```

    )
    else
    {
        mFlowVars->mCurrentLine->children()->add((image);
        image->computeMetrics(mFlowVars->mDefaultWidth);
        mFlowVars->mWidthLeft -= image->getMetrics().width;
        if (mFlowVars->mWidthLeft < 0)
        {
            if (mFlowVars->mCurrentTextState.bits.wordWrap)
                mFlowVars->mWidthLeft = 0;
            else
                mFlowVars->mWidthLeft = mFlowVars->mDefaultWidth;
        }
    }
    delete curIter;
    FlowVars->mCurrentLine->computeMetrics(mFlowVars->mDefaultWidth);
    mFlowVars->mCurrentLine = new WMLLine(smlines, mAlign);
    LineMetrics metrics;
    metrics.baselineOffset = 5;
    metrics.width = mFlowVars->mDefaultWidth;
    mFlowVars->mCurrentLine->setMetrics(metrics);
    delete mFlowVars->mIterStack;
    delete mFlowVars;
    mFlowVars = NULL;
    bits.mFlowDone = true;
    return true;
}

void
WMLParagraph::flowText(WMLTextRun * textRun)
{
    WMLTextRun * newRun;

    mFlowVars->mCurrentTextState = textRun->getState();
    mFlowVars->mCurrentTextState.setWordWrap(bits.misWrapOn);
    textRun->setState(mFlowVars->mCurrentTextState);
    FlowVars->mCurrentTextState.enable();

    if (bits.mUnwrappedSpillover && mFlowVars->mCurrentTextState.bits.wordWrap)
    {
        bits.mUnwrappedSpillover = false;
        mFlowVars->mWidthLeft = 0;
    }

    int cursor = 0;
    bool skipLeadingSpace = mFlowVars->mCurrentLine->children()->getNumElements() == 0;
    while( (newRun = textRun->getNextLine((BLParentGlyph *)mFlowVars->mCurrentLine,
        mFlowVars->mWidthLeft,
        mFlowVars->mDefaultWidth,
        cursor,
        skipLeadingSpace)) != NULL) ||
        textRun->hasMoreText(cursor))
    {
        if (textRun->hasMoreText(cursor) && mFlowVars->mCurrentTextState.bits.wordWrap)
        {
            newLine();
        }
    }
}

else if (textRun->hasMoreText(cursor) && mFlowVars->mCurrentTextState.bits.wordWrap)
{
    mFlowVars->mWidthLeft = mFlowVars->mDefaultWidth;
    bits.mUnwrappedSpillover = true;
}
skipLeadingSpace = true;
}

BLVector<WMLLine *, BLGlyph::kChildrenInc>
WMLLine::sUnusedLines;

BLVector<WMLTextRun *, BLGlyph::kChildrenInc>
WMLTextRun::sUnusedRuns;

WMLLine::WMLLine(BLGlyph * parent, Alignment align)
: WMLLineElement(parent),
  mChildren(0)
{
    mAlign = align;
    mLineWidth = 0;
}

LineMetrics
WMLLine::getMetrics()
{
    BLIterator<BLGlyph *, kChildrenInc> iter(mChildren);
    LineMetrics metrics;
    while(iter.hasNext())
    {
        WMLLineElement * element = dynamic_cast<WMLLineElement *>(iter.getNext());
        LineMetrics elemMetrics = element->getMetrics();
        metrics.width += elemMetrics.width;
        if (elemMetrics.baselineOffset > metrics.baselineOffset)
            metrics.baselineOffset = elemMetrics.baselineOffset;
        if (elemMetrics.overhang > metrics.overhang)
            metrics.overhang = elemMetrics.overhang;
    }
    if (metrics.width > mLineWidth)
        mAlign = kLeft;
    if (mAlign == kCenter)
    {
        metrics.leadingSpace = (mLineWidth - metrics.width) / 2;
        metrics.width += metrics.leadingSpace;
    }
    else if (mAlign == kRight)
    {
        metrics.leadingSpace = mLineWidth - metrics.width; // !!! not +=?
        metrics.width += metrics.leadingSpace;
    }
    return metrics;
}

bool
WMLLine::drawOrClick( const RectangleType & displayArea,
    const Point & offset,
    PointType p,
    bool isDraw)
{
    BLIterator<BLGlyph *, kChildrenInc> iter(mChildren);

```

```

int widthUsed = 0;
Point remainingOffset = offset;
LineMetrics metrics = getMetrics();
widthUsed = metrics.leadingSpace;
if(remainingOffset.x > widthUsed)
{
    remainingOffset.x -= widthUsed;
    widthUsed = 0;
}
else
{
    widthUsed -= remainingOffset.x;
    remainingOffset.x = 0;
}
int index = 0;
while(iter.hasNext())
{
    WMLLineElement * element = dynamic_cast<WMLLineElement*>(iter.getNext());
    LineMetrics elemMetrics = element->getMetrics();
    RectangleType rect = displayArea;
    if(remainingOffset.x > elemMetrics.width)
    {
        remainingOffset.x -= elemMetrics.width;
        continue;
    }
    rect.topLeft.x += widthUsed;
    rect.extent.x -= widthUsed;
    int baseDifference = metrics.baselineOffset - elemMetrics.baselineOffset;
    Point elemOffset = remainingOffset;
    if(elemOffset.y > 0)
    {
        if(baseDifference > elemOffset.y)
        {
            baseDifference -= elemOffset.y;
            elemOffset.y = 0;
        }
        else
        {
            elemOffset.y -= baseDifference;
            baseDifference = 0;
        }
    }
    rect.topLeft.y += baseDifference;
    rect.extent.y -= baseDifference;
    if(!isDraw)
        element->draw(rect, elemOffset);
    else if(!highlightAction == kFindSelection)
    {
        Point pt(p);
        RectangleType compareRect = rect;
        compareRect.topLeft = rect.topLeft;
        compareRect.extent = elemMetrics.getExtent();
        compareRect.extent.x -= elemOffset.x;
        compareRect.extent.y -= elemOffset.y;
        if((compareRect.extent.y > 0) && (compareRect.extent.x > 0))
            if(pt.isInside(compareRect) && element->click(rect, elemOffset, p))

```

```

return true;
}
if( sClickedAction && (!sClickedAction == element->getTextState().action) ||
    (sClickedAction == dynamic_cast<WMLAction*>(element)))
    element->click(rect, elemOffset, p);
}
if(remainingOffset.x > 0)
{
    widthUsed += elemMetrics.width - remainingOffset.x;
    remainingOffset.x = 0;
}
else
{
    widthUsed += elemMetrics.width;
}
if(widthUsed > displayArea.extent.x)
    break;
return false;
}
void
WMLLine::draw(const RectangleType & displayArea, const Point &offset)
{
    drawOrClick(displayArea, offset, Point::sZeroPoint(), true);
}
bool
WMLLine::click(const RectangleType & displayArea,
               const Point &offset,
               PointType p)
{
    return drawOrClick(displayArea, offset, p, false);
}
void *
WMLLine::operator new(size_t size)
{
    if(sUnusedLines.getNumElements() == 0)
        return sMemManager.newChunk(size);
    else
    {
        void * mem = sUnusedLines.removeLast();
        if(sUnusedLines.getNumElements() == 0)
            sUnusedLines.deleteAll();
        return mem;
    }
}
bool
WMLLineElement::click(const RectangleType & displayArea,
                     const Point &offset,
                     PointType /*p*/)
{
    if(mState.action != NULL)
    {
        if(!highlightAction == kFindSelection)
        {
            sClickedAction = mState.action;
            return true;
        }
        else
    }

```

```

    RectangleType rect = getArea(displayArea, offset);
    WinInvertRectangle(&rect, 0);
    ActiveRectVector.add(&rect);
}
return false;
}

Point
WMLLineElement::getExtent()
{
    return getMetrics().getExtent();
}

void
WMLLineElement::mouseSelect()
{
    if(mState.action)
        mState.action->activate();
}

WMLTextRun::WMLTextRun(BLGlyph * parent,
                        CharPtr text,
                        int length,
                        const TextState &state)
: WMLLineElement(parent)
{
    mText = text;
    mLength = length;
    mState = state;
}

WMLTextRun::WMLTextRun(BLGlyph * parent, const WMLTextRun &run)
: WMLLineElement(parent)
{
    mText = run.mText;
    mLength = run.mLength;
    mState = run.mState;
}

void
WMLTextRun::draw(const RectangleType & displayArea, const Point &offset)
{
    if(mLength == 0)
        return;
    mState.enable();
    LineMetrics metrics = getMetrics();
    WinDrawChars( mText, (USHORT) mLength,
                  displayArea.topLeft.x - offset.x,
                  displayArea.topLeft.y - offset.y);
    if(mState.bits.emphasis || mState.bits.italic)
    {
        RectangleType italicArea;
        italicArea.topLeft = displayArea.topLeft;
        italicArea.topLeft.x -= offset.x;
        italicArea.topLeft.y -= offset.y;
        italicArea.extent.x = metrics.width;
        italicArea.extent.y = metrics.height();
        italicize(italicArea);
    }
}

```

```

    WMLTextRun::italicize(const RectangleType & displayArea)
{
    void
    WMLTextRun::italicize(const RectangleType & displayArea)
    {
        int height = displayArea.extent.y;
        RectangleType rect = displayArea;
        RectangleType vacated;

        int shiftAmount = height / kItalicInterval;
        for(int i=0; i < height; i+=kItalicInterval)
        {
            rect.topLeft.y = displayArea.topLeft.y + i;
            rect.extent.y = kItalicInterval;
            WinScrollRectangle(&rect, right, shiftAmount--, &vacated);
            WinEraseRectangle(&vacated, 0);
        }

        WMLTextRun *
        WMLTextRun::getNextLine(BLGlyph * parent,
                                int &pixelsAcross,
                                int &maxSize,
                                int &cursor,
                                bool &skipLeadingSpace)
        {
            if(mText == NULL)
                return NULL;
            if(skipLeadingSpace)
            {
                while(BLUtils::isWhitespace(mText(cursor)) && (cursor < mLength))
                    cursor++;
            }
            if(cursor == mLength)
                return NULL;
            ErrNonFatalDisplayIf(mText == NULL, "mText == NULL");
            LineMetrics metrics = getMetrics();
            WMLTextRun * newRun = NULL;
            Sword pixelsAllowed = pixelsAcross;
            Sword charsInLine = mLength - cursor;
            Boolean fits = false;
            int height = FntBaseLine() + FntDescenderHeight();
            if(BLUtils::isWhitespace(mText + cursor, mLength - cursor))
                fits = true;
            else
            {
                FntCharsInWidth(mText + cursor, &pixelsAllowed, &charsInLine, &fits);
                if(charsInLine == 0)
                    return NULL;
            }
            if(fits)
                charsInLine = mLength - cursor;
            while(!fits || (getTextWidth(mText + cursor, charsInLine, mState, height) >
                           pixelsAcross))
            {
                charsInLine = backupToWhiteSpace(mText + cursor, charsInLine - 1);
                if((charsInLine == 0) && (maxSize == pixelsAcross))
                {
                    // it appears the word is wider than allotted area, so break word
                    // into pieces (you come in peace, you go in pieces)
                    fits = false;
                    pixelsAllowed = pixelsAcross;
                }
            }

```

```

        charsInline = mLength - cursor;
        FntCharsInWidth(mText + cursor, &pixelsAllowed, &charsInline, &state);
        if (charsInline == 0)
            charsInline = 1;
        break;
    }
    else if (charsInline == 0)
        return NULL;
    fits = true;
}

int newCursor = cursor + charsInline;
newRun = new WMLTextRun(parent, *this);
newRun->mText = mText + cursor;
newRun->mLength = charsInline;
pixelsAcross += newRun->getMetrics().width;
cursor = newCursor;
return newRun;
}

int WMLTextRun::backupToWhitespace(CharPtr text, int length)
{
    if (length <= 0)
        return 0;
    int cur = length - 1;
    while ((cur > 0) && !BUtills::isWhitespace(text[cur]))
        cur--;
    if (!BUtills::isWhitespace(text[cur]))
        return 0;
    // now that we found whitespace, skip it too;
    while ((cur > 0) && !BUtills::isWhitespace(text[cur]))
        cur--;
    return cur + 1;
}

WMLTextRun::WMLTextRun(const TextState &state)
{
    mState = state;
    mSrc = NULL;
    mData = NULL;
    mAltText = NULL;
    mAlign = kBottom;
    mParentParagraph = NULL;
    mIsExtentCached = false;
    mTotalExtent.x = mTotalExtent.y = 0;
}

WMLImage::WMLImage(BGLlyph * parent, const WMLImage &original)
{
    mSrc = original.mSrc;
    mAltText = original.mAltText;
    mAlign = original.mAlign;
    mSpace = original.mSpace;
}

```

```

File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM

mState = original.mState;
mData = original.mData;
mParentParagraph = original.mParentParagraph;
misExtentCached = false;
}

bool
WMLImage::isDownloaded()
{
    return mData != NULL;
}

void
WMLImage::computeMetrics(int /*lineWidth*/)
{
    Point extent = getExtent();
    mState.enable();
    mMetrics.width = extent.x;
    { (mAlign == kTop)
        mMetrics.baselineOffset = min(PntBaseline(), extent.y);
        mMetrics.overhang = extent.y - mMetrics.baselineOffset;
    }
    else if (mAlign == kCenter) || (mAlign == kMiddle)
    {
        mMetrics.baselineOffset = extent.y / 2;
        mMetrics.overhang = extent.y - mMetrics.baselineOffset;
    }
    else if (mAlign == kBottom)
    {
        mMetrics.baselineOffset = extent.y;
        mMetrics.overhang = 0;
    }
}

void
WMLImage::draw(const RectanglType & displayArea, const Point &offset)
{
    if (mData == NULL)
        return; // !!! throw an exception, someday
    else
    {
        WinDrawBitmap( mData,
            displayArea.topLeft.x - offset.x,
            displayArea.topLeft.y - offset.y);
    }
}

void
WMLImage::setImageData(CharPtr data)
{
    mData = (BitmapPtr) data;
    if (mParentParagraph != NULL)
        mParentParagraph->needsReflow();
}

Point
WMLImage::getExtent()
{
    Point extent(0, 0);
    if (mData == NULL)
        return extent;
}

if (misExtentCached)
    return misExtent;
bool enough = false;
extent.x = (Short) mData->width;
extent.y = (Short) mData->height;
mTotalExtent = extent;
misExtentCached = true;
return extent;
}

// BGLyph.h
#pragma once
#include "BLTypes.h"
#include "BLVector.h"
#include "BLIterator.h"
#include "BLMemoryManager.h"
#include "WMLState.h"

class BLWMLSession;
class WMLAction;
class WMLTemplate;
class WMLDoc;
class WMLTimer;

class BGLyph
{
public:
    enum { kChildrenInc = 4 };
    enum Alignment { kTop, kBottom, kCenter, kLeft, kRight, kMiddle };
    enum HitTestAction { kFindSelection, kHitTest, kUnhighlight };

    virtual bool click( const RectanglType & displayArea,
        const Point &offset,
        PointType p);
    virtual void draw( const RectanglType & displayArea,
        const Point &offset);
    bool drawOrClick(const RectanglType & displayArea,
        const Point &offset,
        PointType p,
        bool isDraw);
    virtual void mouseSelect() {}
    virtual Point getExtent() { return getExtent(Point::sZeroPoint()); }
    virtual Point getExtent(const Point &areaExtent);
    RectanglType getBounds(const Point &topLeft);
    RectanglType getArea(const RectanglType & displayArea,
        const Point &offset);
    void * operator new(size_t);
    void operator delete(void *, size_t);
    static void * newChunk(unsigned long size) { return sMemManager.newChunk(size); }

    virtual TextState getTextState() { return TextState::getBasicState(); }
    virtual BLVector<BGLyph> * kChildrenInc() { return children(); }
    { return NULL; }

    static void flushAll();
    static void setSession(BLWMLSession * session) { sSession = session; }
    static void setMemoryManagers();
    static WMLAction * sClickedAction;
    static BLVector<RectanglType, 4> sActiveRectVector;
}

```



```

static HILiteAction  mHiliteAction;
friend class WBXMLParser;

protected:
    BGLGlyph(BGLGlyph * parent);
    BGLGlyph() {}

    virtual bool    intersects(Point topLeft,
                               PointType p);

    static BLWMLSession * getSession() { return sSession; }

    static CharPtr   cloneString(CharPtr str);

    static BLMemoryManager sMemManager;
    static BLWMLSession * sSession;

};

class WMLIntrinsicEvents
{
public:
    WMLIntrinsicEvents()
    {
        mOnEnterForward = mOnEnterBackward = mOnTimer = mOnPick = NULL;
    }

    WMLAction * mOnEnterForward;
    WMLAction * mOnEnterBackward;
    WMLAction * mOnTimer;
    WMLAction * mOnPick;

};

class BLParentGlyph : public BGLGlyph
{
public:
    virtual BLVector<BGLGlyph*, kChildrenInc> * children()
    { return &mChildren; }

protected:
    BLParentGlyph(BGLGlyph * parent = NULL): BGLGlyph(parent),
        mChildren(0) {}

    BLParentGlyph() : mChildren(0) {}

    BLVector<BGLGlyph*, kChildrenInc> mChildren;

};

class WMLRoot;

class WMLCard : public BLParentGlyph
{
public:
    WMLCard();
    WMLCard(BGLGlyph * parent, const WMLCard & original, WMLRoot *
root);

    virtual void    draw(const RectangleType & displayArea, const Point &offset);
    virtual void    flow(const Point &areaExtent);

    void    addAction(WMLDo * doP);

    BLVector<WMLDo*, kChildrenInc> &getNamedActions() { return mNamedActions; }
    WMLIntrinsicEvents &getIntrinsicEvents() { return mEvents; }

    void    activate();
    void    deactivate();
    void    setTimer(WMLTimer * timer) { mTimer = timer; }

```

```

doneParsing();

onEnterEvent(NavigationAction lastAction);

WBXMLParser;
WMLRoot;

protected:
    WMLIntrinsicEvents mEvents;
    BLVector<WMLDo*, kChildrenInc> mNamedActions;

    CharPtr    mID;
    CharPtr    mTitle;

    WMLTimer * mTimer;

    bool    mIsTitleSet;
    bool    mIsOrdered;
    bool    mNewContext;

};

class WMLRoot : public BLParentGlyph
{
public:
    WMLRoot();

    virtual void    draw(const RectangleType & displayArea, const Point &offset);
    virtual bool    click(const RectangleType & displayArea,
                           const Point &offset,
                           PointType p);

    virtual Point    getExtent(const Point &areaExtent);

    friend class    WBXMLParser;

    void    addAction(WMLDo * doP);

    BLVector<WMLDo*, kChildrenInc> &getNamedActions() { return mNamedActions; }

    bool    gotoCard(CharPtr cardID);

    CharPtr    getActiveCardID() { if(mActiveCard) return mActiveCard->mID;
                                   else return NULL; }

    void    onEnterEvent(NavigationAction lastAction);

protected:
    WMLCard *    mDefaultCard;
    WMLCard *    mActiveCard;
    WMLTemplate * mTemplate;

    CharPtr    mDomainAccess;
    CharPtr    mPathAccess;

    BLVector<WMLDo*, kChildrenInc> mNamedActions;

};

class WMLGlyphContainer : public BLParentGlyph
{
public:
    WMLGlyphContainer() : BLParentGlyph(NULL) {}

};

class WMLLineElement : public BGLGlyph
{
public:
    WMLLineElement(BGLGlyph * parent) : BGLGlyph(parent) {}
    WMLLineElement() : BGLGlyph(NULL) {}

```

```

class WMLParagraph : public BLParentGlyph
{
public:
    WMLParagraph(BLGlyph * parent = NULL);
    WMLParagraph(BLGlyph * parent, const WMLParagraph &original);

    virtual void
    setMetrics(const LineMetrics & metrics) { }

    virtual void
    computeMetrics(int /*lineWidth*/) { }

    virtual bool
    click(const RectangleType & displayArea,
          const Point &offset,
          PointType p);

    virtual void
    mouseSelect();

    virtual TextState
    getTextState() { return mState; }

protected:
    TextState mState;

    WMLLine(BLGlyph * parent, Alignment align);

    bool
    drawOrClick(const RectangleType & displayArea,
                const Point &offset,
                PointType p,
                bool isDraw);

    virtual void
    draw(const RectangleType & displayArea, const Point &offset);

    virtual LineMetrics getMetrics();

    virtual void
    computeMetrics(int lineWidth) { mLineWidth = lineWidth; }

    virtual bool
    click(const RectangleType & displayArea,
           const Point &offset,
           PointType p);

    void *
    operator new(size_t);

    virtual BLVector<BLGlyph*, kChildrenInc> * children()
    { return &mChildren; }

    static BLVector<WMLLine *, kChildrenInc> sUnusedLines;

protected:
    BLVector<BLGlyph*, kChildrenInc> mChildren;

    int
    mLineWidth;
    Alignment mAlign;

    class WMLTextRun;
    class ParagraphFlowVars;

public:
    ParagraphFlowVars() { mIterStack = NULL; mCurrentLine = NULL; }

    // incremental flow variables
    BLStack * mIterStack;
    WMLLine * mCurrentLine;
    TextState mCurrentTextState;
    int mWidthLeft;
    int mDefaultWidth;

    WMLTextRun * getNextLine(BLGlyph * parent,
                              int &pixelsAcross,
                              int &maxSize,
                              int &cursor,
                              bool skipLeadingSpace);

    bool
    hasMoreText(int cursor) { return cursor < mLength; }

    TextState &
    getState() { return mState; }

    void
    setState(const TextState &state) { mState = state; }

    int
    findLongestWord();
};

```

```

        void * operator new(size_t)
    static BLVector<WMLTextRun *, kChildrenInc> sUnusedRuns;

protected:
    int backupToWhitespace(CharPtr text, int length);
    void italicize(const RectangleType & displayArea);
    CharPtr mText;
    int mLength;

class WMLBreak : public BLGlyph
{
public:
    WMLBreak(BLGlyph * parent) : BLGlyph(parent) {}

};

class WMLImage : public WMLLineElement
{
public:
    WMLImage(const TextState &state);
    WMLImage(BLGlyph * parent, const WMLImage &original);
    virtual void draw(const RectangleType & displayArea, const Point &offset);
    virtual Point getExtent();
    virtual void computeMetrics(int lineWidth);
    virtual LineMetrics getMetrics() { return mMetrics; }

    bool isDownloaded();
    CharPtr getAltText() { return mAltText; }
    CharPtr getSrc() { return mSrc; }
    void setImageData(CharPtr data);

friend class WBXMLParser;

protected:
    LineMetrics mMetrics;
    CharPtr mSrc;
    CharPtr mAltText;
    Point mSpace;
    Point mTotalExtent;
    BimapPtr mData;
    Alignment mAlign;
    WMLParagraph * mParentParagraph;

    bool mIsExtentCached;
};

// BLIterator.h
#include "BLVector.h"
#include "BLStack.h"
#pragma once

template <class T, int increment_>
class BLIterator
{
public:
    BLIterator(const BLVector<T, increment_> &vector);
    hasNext() const { return mCurrentPos < mVector.getNumElements(); }
    getNext() { return mElements[mCurrentPos++]; }
};

```

```

MemoryBlock * block = NULL;
// align on even boundary:
if((size % 2) == 1)
    size++;
if(numElems > 0)
{
    block = mPool.getElementAt(numElems - 1);
    if(size > block->bytesFree())
    {
        size_t newChunkSize = max(block->physSize + kElemSize, block->logSize + size);
        #ifdef USE_PALM_MEM
            size_t newChunkSize = max(block->physSize + kElemSize, block->logSize + size);
        #endif
        Err err = MemPtrResize(block, newChunkSize + block->getOverhead());
        if(err)
        {
            // can't make it bigger, so shrink it to size of data
            MemPtrResize(block, block->logSize + block->getOverhead());
            block->physSize = block->logSize;
            block = NULL;
        }
        else
        {
            block->physSize = (UShort) newChunkSize;
        }
        block = NULL;
    }
    #else
        block = NULL;
    #endif
}
if(block == NULL)
{
    size_t chunkSize = max(size, kElemSize);
    #ifdef USE_PALM_MEM
        block = (MemoryBlock *) MemPtrNew(chunkSize + block->getOverhead());
        // reportFreeMem();
    #else
        block = (MemoryBlock *) new UChar(chunkSize + block->getOverhead());
    #endif
    if(block == NULL)
        throw OutOfMemoryException();
    block->physSize = (UShort) chunkSize;
    block->logSize = 0;
    mPool.add(block);
}
void * chunkStart = (UChar *)block->memory + block->logSize;
block->logSize += (UShort) size;
return chunkStart;
}

void
BLMemoryManager::reportFreeMem()
{
    Char buf[100];
    ULONG freeBytes, maxP;
    MemHeapFreeBytes(0, &freeBytes, &maxP);
    StrPrintf(buf, "Mem Free: %li, Biggest Chunk: %li, freeBytes, maxP);
    FrmCustomAlert(CustomAlert, buf, "", "");
}
// BLMemoryManager.h

#pragma once
#include "size_t.h"
#include "BLVectorHeader.h"

class MemoryBlock
{
public:
    static unsigned long getOverhead() { return sizeof(MemoryBlock) - sizeof(UChar); }
    static unsigned long bytesFree() { return physSize - logSize; }

    UShort logSize;
    UShort physSize;
    UChar memory;
};

class BLMemoryManager
{
public:
    enum { kMemInitLen = 2, kMemInc = 2, kElemSize = 2048; }

    BLMemoryManager();
    ~BLMemoryManager();

    void * newChunk(unsigned long);
    void flushAll();
    static void reportFreeMem();

protected:
    BLVector<MemoryBlock *, kMemInc> mPool;
}; // BLMemoryManager.cpp

#ifdef ENABLE_NETWORKING
#include "BLNetworking.h"
#include "BLUtils.h"
#include "BLForm.h"
#include "BLPrefs.h"
#undef memcpy
#include <sys_socket.h>

const DWORD kMaxResultSize = 32*1024;
const int kBufferSize = 1024;
const int kMaxURLLength = 128;
const int kPageTimeout = 20;
const int kSendDelay = 2;

extern Word AppNetRefnum;
extern Long AppNetTimeout;

BLNetworking::BLNetworking()
{
    mReader = NULL;
    mBuffer = NULL;
    mRequestP = NULL;
    mTimerP = NULL;
    mGatewayIP = NULL;
    mContentLocation = NULL;
    mNetLibH = NULL;
    mInetSockH = NULL;

    mLastStatus = inetStatusPrvInvalid;
    mLibType = kNoLib;
    mIsNetLibWaitingForData = false;
    mTimeoutVal = kSendDelay * SysTicksPerSecond();
}

BLNetworking::~BLNetworking()

```

```

    }
    }

    Err
    BLNNetworking::start()
    {
        DWord value;
        mBytesUsed = 0;
        Err err = FtrGet(netFtrCreator, netFtrNumVersion, &value);
        if(!err) && (value != 0)
        {
            if(err = SysLibFind("Net.lib", &AppNetRefNum)) != 0)
                goto INET_OPEN;
            DWord ifCreator=0;
            Word ifInstance=0;
            if((err = NetLibIFGet(AppNetRefNum, 0, &ifCreator, &ifInstance)) != 0)
                goto INET_OPEN;
            Word ifErrs;
            err = NetLibOpen(AppNetRefNum, &ifErrs);
            if(err == netErrAlreadyOpen)
            {
                err = 0;
                ifErrs = 0;
            }
            if(!err && !ifErrs)
            {
                AppNetTimeout = kPageTimeout*SysTicksPerSecond();
                mTimeoutVal = kSendDelay + SysTicksPerSecond();
                mLibType = kNetLib;
                return 0;
            }
            else
            {
                NetLibClose(AppNetRefNum, false);
            }
        }
        INET_OPEN:
        r = FtrGet(inetLibFtrCreator, inetFtrNumVersion, &value);
        if(!err) && (value != 0)
        {
            Word config;
            if((err = SysLibFind("Inet.lib", &inetLibRefNum)) != 0)
                return err;
            if((err = INetLibConfigIndexFromName(
                mInetLibRefNum,
                (INetConfigNamePtr) inetCfgNameCTPDefault,
                &config)) != 0)
                return err;
            if((err = INetLibOpen(
                mInetLibRefNum,
                config,
                0,
                NULL,
                0,
                &inetLibH))
                != 0)
                return err;
            cache
        }
    }

```

```

    DWord conv = ctpConvNone;
    return 0;
}

// turn off compression
if((err = INetLibSettingSet(mInetLibRefNum,
    mInetLibH,
    inetSettingConvAlgorithm,
    &conv,
    sizeof(conv))) != 0)
    return err;

DWord maxResultSize = kMaxResultSize;
// set max size:
if((err = INetLibSettingSet(mInetLibRefNum,
    mInetLibH,
    inetSettingMaxRepSize,
    &maxResultSize,
    sizeof(conv))) != 0)
    return err;

mLibType = kInetLib;
mBuffer = new Char[kBufferSize];
mContentLocation = new Char[kMaxURLLength];
return 0; // INetLib found, return
}

ErrDisplay("Cannot open NetLib");
return -1;
}

Err
BLNNetworking::stop()
{
    Err err = 0;
    if(mLibType == kInetLib)
    {
        err = INetLibClose(mInetLibRefNum, mInetLibH);
        delete [] mBuffer;
        mBuffer = NULL;
        delete [] mContentLocation;
        mContentLocation = NULL;
    }
    else if(mLibType == kNetLib)
    {
        if(mRequestP)
        {
            delete mRequestP;
            mRequestP = NULL;
        }
        delete [] mGatewayIP;
        mGatewayIP = NULL;
        err = NetLibClose(AppNetRefNum, false);
    }
    return err;
}

void
BLNNetworking::setTimer(ULong timeout, Timer * timerP)
{
    mTimerP = timerP;
    mTimeoutVal = (SDWord) min(timeout/2, mTimeoutVal);
}

```

```

void
BLNetworking::clearTimer()
{
    mTimerP = NULL;
    mTimeoutVal = kSendDelay * SysTicksPerSecond();
}

void
BLNetworking::getEvent(EventType & event)
{
    do {
        if (mLibType == kINetLib)
        {
            INetLibGetEvent(minetLibRefNum,
                            minetLibH,
                            (INetEventType*) &event,
                            mTimeoutVal);

            if (mTimerP && mTimerP->checkTime())
            {
                mTimeoutVal = kSendDelay * SysTicksPerSecond();
                mTimerP = NULL;
            }

            if (event.eType != nilEvent)
                return;
            else if (mLastStatus == inetStatusSendingRequest)
                displayStatus(inetStatusWaitingForResponse);
            continue;
        }
        else if (mIsNetLibWaitingForData)
        {
            fd_set arfds, awfds, rfd, wfd;
            int nfd = 0;

            if (mRequestP == NULL)
            {
                mIsNetLibWaitingForData = false;
                continue;
            }

            int sock = mRequestP->getSock();

            FD_ZERO(&arfds);
            FD_ZERO(&awfds);
            FD_SET(sock, &arfds);
            FD_SET(STDIN_FILENO, &arfds);
            nfd = sock + 1;

            while (true)
            {
                bcopy(&arfds, &rfd, sizeof(rfd));
                bcopy(&awfds, &wfd, sizeof(wfd));

                if (select(nfd, &rfd, &wfd, (fd_set*)0, (struct timeval*)0) < 0)
                {
                    EvtGetEvent(&event, 1);
                    return;
                }

                bool nothingSet = true;

                if (FD_ISSET(sock, &rfd))
                {
                    nothingSet = false;
                    if (!mParsedHeader)
                    {
                        if (mRequestP->parseHTTPHeader())
                        {
                            mParsedHeader = true;
                        }
                        else
                    }
                }
            }
        }
        else if (mIsNetLibWaitingForData == false)
        {
            mLastStatus = inetStatusPrvInvalid;

            if (mReader->readData(NULL, 0) == 0)
            {
                mIsNetLibWaitingForData = false;
                break;
            }

            if (FD_ISSET(STDIN_FILENO, &rfd))
            {
                nothingSet = false;
                EvtGetEvent(&event, 1);
                return;
            }

            if (nothingSet)
                break;
        }

        EvtGetEvent(&event, mTimeoutVal);

        if (mTimerP && mTimerP->checkTime())
        {
            mTimeoutVal = kSendDelay * SysTicksPerSecond();
            mTimerP = NULL;
        }

        if (event.eType != nilEvent)
            return;
    } while (true);
}

void
BLNetworking::dataReady(Handle sockH, Word status)
{
    // !!! someday, enable multiple sockets (up to 4 on with NetLib)

    if (mLibType == kINetLib)
    {
        ULONG numAvail;

        displayStatus(status);

        Err err = INetLibSockRead( minetLibRefNum,
                                   sockH,
                                   mBuffer + mBytesUsed,
                                   kBufferSize - mBytesUsed,
                                   &numAvail,
                                   kPageTimeout * SysTicksPerSecond());

        if (!err && (numAvail > 0))
        {
            if (mReader)
            {
                ULONG bytesRead = mReader->readData(mBuffer, numAvail + mBytesUsed);
                ULONG bytesLeft = mBytesUsed + numAvail - bytesRead;

                if (bytesLeft > 0)
                {
                    MemMove(mBuffer, mBuffer + bytesRead, bytesLeft);
                    mBytesUsed = (UINT) bytesRead;
                }
            }
        }
    }
}

```

```

    else
    {
        mBytesUsed = 0;
    }
}
else
{
    mReader->readData(NULL, 0);
    INetLibSockClose(mInetLibRefNum, sockH);
    if(mInetSockH == sockH)
        mInetSockH = NULL;
}
}

int
BINetworking::peek(CharPtr buffer, int length)
{
    turn recv(getSock(), buffer, (word) length, netIOFlagPeek);
}

int
BINetworking::readData(CharPtr buffer, int length)
{
    return NetLibReceive(AppNetRefNum, getSock(), buffer, (word) length, 0, 0, 0, &errno);
}

CharPtr
BINetworking::openURL(CharPtr url, bool updateStatus)
{
    Err err = 0;

    CharPtr gatewayP = BLPrefs::sPrefs.mServerAddress;

    if(mLibType == kNetLib)
    {
        CharPtr fullURL = new Char(strlen("http://") + strlen(gatewayP) + strlen(url) + 1);

        if(fullURL == NULL)
            return "Out of memory."; // !!! Do something smart here.

        strcpy(fullURL, gatewayP);
        strcpy(fullURL + strlen(fullURL), url);

        // !!! delete url if error

        if((err = INetLibSockOpen(mInetLibRefNum,
            mInetLibH,
            inetSchemeHTTP,
            &mInetSockH)) != 0)
            return "Could not open internet socket";

        if((err = INetLibSockConnect(mInetLibRefNum,
            mInetSockH,
            (BytePtr) gatewayP,
            kHTTPPort,
            -1)) != 0)
            return "Could not connect to Gateway";

        if((err = INetLibSockHTTPReqCreate(mInetLibRefNum,
            mInetSockH, "GET",
            (BytePtr) fullURL,
            NULL)) != 0)
            return "Could not get URL";

        if((err = INetLibSockHTTPReqSend(mInetLibRefNum,
            mInetSockH, NULL, 0,
            -1)) != 0)
            return "Could not send HTTP request";

        // alternative:
        // Err err = INetLibURLOpen(mInetLibRefNum, mInetLibH, (BytePtr)url, NULL,
        // kPageTimeout + SysTicksPerSecond(), 0);

        &mInetSockH,
        //
        }
        else if(mLibType == kNetLib)
        {
            if(mRequestP && (mRequestP->requestFile(url) > 0))
            {
                if(updateStatus)
                    displayStatus(inetStatusWaitingForResponse);
                mIsNetLibWaitingForData = true;
                mParsedHeader = false;
                return NULL;
            }
            if(mRequestP)
            {
                delete mRequestP;
                mRequestP = NULL;
            }
            if(updateStatus)
                displayStatus(inetStatusConnecting);
            if(mGatewayIP == NULL)
            {
                struct hostent *hp;
                if((hp = gethostbyname(gatewayP)) == NULL)
                {
                    Byte allInterfacesUp;
                    Word inetErr;
                    Err err = NetLibConnectionRefresh(AppNetRefNum, true, &allInterfacesUp,
                        &inetErr);

                    if(!err && !inetErr)
                    {
                        if((hp = gethostbyname(gatewayP)) == NULL)
                        {
                            mIsNetLibWaitingForData = false;
                            return "Cannot send request";
                        }
                    }
                    else
                    {
                        mIsNetLibWaitingForData = false;
                        return "Cannot send request";
                    }
                }
                mGatewayIP = BLUtils::IPToString(*Ulong*) hp->h_addr_list[0];
            }
            int sock = NetUTCPopen(mGatewayIP, NULL, kHTTPPort);
            if(sock < 0)
            {
                Byte allInterfacesUp;
                Word inetErr;
                Err err = NetLibConnectionRefresh(AppNetRefNum, true, &allInterfacesUp,
                    &inetErr);
                if(!err && !inetErr)
                {
                    sock = NetUTCPopen(mGatewayIP, NULL, kHTTPPort);
                }
            }
        }
    }
}

```

```

    case inetStatusClosed: // closed
        break;
    if (sock >= 0)
    {
        mRequestP = new HTTPRequest(sock, gatewayP);
        if (mRequestP->requestFile(uri) > 0)
        {
            if (updateStatus)
                displayStatus(inetStatusWaitingForResponse);

            mIsNetLibWaitingForData = true;
            mParsedHeader = false;
            return NULL;
        }
        else return "Cannot send request";
    }
    else
    {
        mIsNetLibWaitingForData = false;
        return "Cannot open socket";
    }
}

urn NULL;

void
BLNetworking::cancelOpenURL()
{
    if (mLibType == kNetLib)
    {
        if (mInetSockH != NULL)
        {
            INetLibSockClose(mInetLibRefNum, mInetSockH);
            mInetSockH = NULL;
        }
    }
    else
    {
        mIsNetLibWaitingForData = false;
        delete mRequestP;
        mRequestP = NULL;
    }
}

void
BLNetworking::displayStatus(Word status)
{
    if ((status != mLastStatus) && (status != inetStatusPrvInvalid))
    {
        CharPtr message = NULL;

        switch(status)
        {
            case inetStatusNew: // just opened
            case inetStatusResolvingName: // looking up host address
            case inetStatusNameResolved: // found host address
            case inetStatusConnecting: // connecting to host
            case inetStatusConnected: // connected to host
                message = "Connecting...";
                break;
            case inetStatusSendingRequest: // sending request
                message = "Sending...";
                break;
            case inetStatusWaitingForResponse: // waiting for response
                message = "Waiting...";
                break;
            case inetStatusReceivingResponse: // receiving response
            case inetStatusResponseReceived: // response received
            case inetStatusClosingConnection: // closing connection
                break;
        }

        if (message != NULL)
        {
            if (length > 0)
                return mContentLocation;
            else
                return mContentLocation;
        }
    }
}

int
BLNetworking::getContentLength()
{
    if (mRequestP)
    {
        if (mRequestP->getEncodingType() == HTTPRequest::kRaw)
            return mRequestP->getContentLength();
        else
            return -1;
    }
    else return -1;
}

CharPtr
BLNetworking::getContentLocation()
{
    if (mLibType == kNetLib)
    {
        if (mInetSockH != NULL)
        {
            Word setting = inetSockSettingURL;
            Word length = kMaxURLLength;

            INetLibSockSettingGet(mInetLibRefNum, mInetSockH, setting, mContentLocation,
                                &length);
        }
    }
    else
    {
        if (length > 0)
            return mContentLocation;
        else
            return mContentLocation;
    }
}

Byte buffer[16];

INetLibSockSettingGet(mInetLibRefNum, mInetSockH, setting, buffer, sizeof(buffer));

(INetContentTypeEnum)*/
    return kWMILC;
}

if (mRequestP)
{
    CharPtr contentType = mRequestP->getContentType();

    if (contentType == NULL)
        return kUnknown;
    else if (StrCaselessCompare(contentType, "text/vnd.wap.wmlc") == 0)
        return kWMILC;
    else if (StrCaselessCompare(contentType, "image/vnd.palm.pbmp") == 0)
        return kPalmBitmap;
    return kUnknown;
}

int
BLNetworking::getContentLength()
{
    if (mRequestP)
    {
        if (mRequestP->getEncodingType() == HTTPRequest::kRaw)
            return mRequestP->getContentLength();
        else
            return -1;
    }
    else return -1;
}

CharPtr
BLNetworking::getContentLocation()
{
    if (mLibType == kNetLib)
    {
        if (mInetSockH != NULL)
        {
            Word setting = inetSockSettingURL;
            Word length = kMaxURLLength;

            INetLibSockSettingGet(mInetLibRefNum, mInetSockH, setting, mContentLocation,
                                &length);
        }
    }
    else
    {
        if (length > 0)
            return mContentLocation;
        else
            return mContentLocation;
    }
}

```



```

    return NULL;
}
else return NULL;
}

if(mRequestP)
    return mRequestP->getContentLocation();
else return NULL;
}

#endif// BLNetworking.h

#pragma once

#include "BLTypes.h"
#include "HTTPRequest.h"

class BLNetworking
{
public:
    enum NetLibType { kNetLib, kNetLib, kNetLib };
    enum ContentType { kUnknown, kWMLC, kPalmBitmap };

    BLNetworking();
    ~BLNetworking();

    start();
    stop();

    void    getEvent(EventType & event);
    void    dataReady(Handle sockH, Word status = inetStatusPrvInvalid);

    int     peek(CharPtr buffer, int length);
    int     readData(CharPtr buffer, int length);

    void    registerReader(DataReader * reader)
    { mReader = reader; }

    CharPtr    openURL(CharPtr url, bool updateStatus = true);
    void        cancelOpenURL();
    void        displayStatus(Word status);

    NetLibType    getNetLibType() { return mLibType; }

    int     getSock() { if(mRequestP) return mRequestP->getSock();
                      else return -1; }

    ContentType    getContentType();

    int     getContentLength();

    HTTPRequest::EncodingType    getEncodingType() { if(mRequestP) return
mRequestP->getEncodingType();
    HTTPRequest::kUnknown; }

    CharPtr    getContentLocation();

    void        setTimer(ULong timeout, Timer * timerP);
    void        clearTimer();

protected:
    DataReader *    mReader;
    char *          mBuffer;
    UInt           mBytesUsed;

```

```

    UInt           mNetLibRefNum;
    Handle         mNetLibH;
    minetSockH;

    Word          mLastStatus;
    bool          mIsSending;

    HTTPRequest * mRequestP;

    NetLibType    mLibType;
    bool          mIsNetLibWaitingForData;
    bool          mParsedHeader;

    Timer *       mTimerP;
    SDWord        mTimeoutVal;

    CharPtr       mGatewayIP;
    CharPtr       mContentLocation;

    // BLPagesDB.cpp
#include "BLPagesDB.h"
#include "BrowserApp.h"

void
BLPagesDB::open()
{
    BLDatabase::open(BrowserApp::appFileCreator, 'docs', "BLDocs", dmModeReadOnly);

    mOpenPage = NULL;
}

void
BLPagesDB::close()
{
    if(mOpenPage != NULL)
    {
        MemHandleUnlock(mOpenPage);
        DmReleaseRecord(mRef, mOpenPageNum, false);
        mOpenPage = NULL;
    }

    BLDatabase::close();
}

RawPageData *
BLPagesDB::get(CharPtr url)
{
    UInt i=0;

    if(mOpenPage != NULL)
    {
        MemHandleUnlock(mOpenPage);
        DmReleaseRecord(mRef, mOpenPageNum, false);
        mOpenPage = NULL;
    }

    UInt numRecords = DmNumRecords(mRef);
    while(i < numRecords)
    {
        BLPageHand pageHand = (BLPageHand) DmGetRecord(mRef, i++);

        if(pageHand == NULL)
            continue;

        BLPagePtr pageP = (BLPagePtr) MemHandleLock(pageHand);

        if(StrCompare(url, &pageP->data) == 0)
        {
            mOpenPage = pageHand;
            mOpenPageNum = i-1;

            return getPageData(pageHand, pageP);
        }
    }
}

```

```

        BLPPageHand mOpenPage;
        BLPPageHand mOpenPageNum;
    }; // BLPrefs.cpp

```

```

#include "BLPrefs.h"
#include "BrowserApp.h"

```

```

PrefsType
BLPrefs::sPrefs;

```

```

void
BLPrefs::read()

```

```

{
    Word prefsSize;

```

```

    // Read the saved preferences / saved-state information.
    prefsSize = sizeof(sPrefs);
    if (PrefGetAppPreferences(BrowserApp::appFileCreator, 0, &sPrefs, &prefsSize, true) !=
        noPreferenceFound)
    {

```

```

    }
    else
    {

```

```

        // set default prefs:
        StrCopy(sPrefs.mServerAddress, KHTTPGateway);
        sPrefs.mShowImages = true;
    }

```

```

}

```

```

void
BLPrefs::write()

```

```

{
    // Write the saved preferences / saved-state information. This data
    // will be backed up during a HotSync.
    PrefSetAppPreferences( BrowserApp::appFileCreator, 0,
        BrowserApp::kPrefsVersion,
        &sPrefs, sizeof(sPrefs), true);
}

```

```

/// BLPrefs.h

```

```

#pragma once

```

```

class PrefsType

```

```

{
    public:

```

```

        bool        mShowImages;
        Char        mServerAddress[64];

```

```

};

```

```

class BLPrefs

```

```

{
    public:

```

```

        static void    read();
        static void    write();

```

```

        static PrefsType sPrefs;

```

```

}; // BLPrefs.h

```

```

#pragma once

```

```

class BLNode

```

```

{
    public:

```

```

        BLNode * mNext;

```

```

};

```

```

class BLStack

```

```

{
    public:

```

```

        BLStack() { mTop = NULL; }
        ~BLStack() {}
        virtual

```

```

    }
    MemHandleUnlock(pageHand);
    DmReleaseRecord(mRef, i-1, false);
}

```

```

// no page found in cache

```

```

return NULL;

```

```

BLPageHand pageHand = (BLPageHand) DmGetResource('tSTR', 1000);

```

```

RawPageData * pageData = new RawPageData(false, false, pageHand);

```

```

pageData->mData = (CharPtr) MemHandleLock(pageHand);

```

```

pageData->mLength = (UInt) MemHandleSize(pageHand);

```

```

// Don't include NULL terminator in length

```

```

if (pageData->mLength > 0)

```

```

    pageData->mLength--;

```

```

    return pageData;

```

```

}

```

```

RawPageData *
BLPagesDB::getPageData(BLPageHand pageHand, BLPagePtr pageP)

```

```

{
    RawPageData * pageData = new RawPageData(false, false, pageHand);

```

```

    UInt offset = StrLen(sPageP->data) + 1;

```

```

    pageData->mData = sPageP->data + offset;

```

```

    pageData->mLength = (UInt) MemPtrSize(pageP) - offset - (UInt) sizeof(UIntLong);

```

```

    if (((UIntLong) pageData->mData % 2) != 0)
    {

```

```

        pageData->mData++;

```

```

        pageData->mLength--;

```

```

    }

```

```

    return pageData;

```

```

}

```

```

// BLPagesDB.h

```

```

#pragma once

```

```

#ifdef "BLDatabase.h"
#endif

```

```

struct BLPage

```

```

{
    ULong reserved;

```

```

    Char data;

```

```

};

```

```

typedef BLPage * BLPagePtr;

```

```

typedef BLPagePtr * BLPageHand;

```

```

class BLPagesDB : public BLDatabase

```

```

{
    public:

```

```

        virtual void    open();

```

```

        virtual void    close();

```

```

        RawPageData * get(CharPtr url);

```

```

protected:

```

```

        RawPageData * getPageData(BLPageHand pageHand, BLPagePtr pageP);

```

```

void push(BINode * item)
{
    item->mNext = mTop;
    mTop = item;
}

BINode * pop()
{
    BINode * poppedNode = mTop;
    if(mTop != NULL)
    {
        mTop = mTop->mNext;
        poppedNode->mNext = NULL;
    }
    return poppedNode;
}

protected:
    BINode * top() const { return mTop; }

    BINode * mTop;

}; #include "BLTextTags.h"

BLTextTags::BLTextTags(BLGlyph * parent, XMLElement * elem)
: BLGlyph(parent, false)
{
}

BLTextTags::~BLTextTags()
{
}

bool
BLTextTags::draw(const RectangleType & displayArea, const Point offset)
{
    return BLGlyph::draw(displayArea, offset);
}

Point
BLTextTags::getExtent(const RectangleType & displayArea, const Point offset)
{
    return BLGlyph::getExtent(displayArea, offset);
}

b
BLTextTags::registerIfTextTag(BLGlyph * glyphP, XMLElement * child)
{
    bool handled = false;
    if (StrCompare(child->mName, "em") == 0)
    {
        WMLEmphasis * emphasis = new WMLEmphasis(glyphP, child);
        emphasis->layoutNode(child);
        handled = true;
    }
    else if (StrCompare(child->mName, "strong") == 0)
    {
        WMLStrong * strong = new WMLStrong(glyphP, child);
        strong->layoutNode(child);
        handled = true;
    }
    else if (StrCompare(child->mName, "i") == 0)
    {
        WMLUnderline * underline = new WMLUnderline(glyphP, child);
        underline->layoutNode(child);
        handled = true;
    }
    else if (StrCompare(child->mName, "u") == 0)
    {
        WMLUnderline * underline = new WMLUnderline(glyphP, child);
        underline->layoutNode(child);
        handled = true;
    }
    else if (StrCompare(child->mName, "big") == 0)
    {
        WMLBig * big = new WMLBig(glyphP, child);
        big->layoutNode(child);
        handled = true;
    }
    else if (StrCompare(child->mName, "small") == 0)
    {
        WMLSmall * small = new WMLSmall(glyphP, child);
        small->layoutNode(child);
        handled = true;
    }
    return handled;
}

#pragma mark -----
WMLUnderline::WMLUnderline(BLGlyph * parent, XMLElement * elem)
: BLTextTags(parent, elem)
{
}

WMLUnderline::~WMLUnderline()
{
}

bool
WMLUnderline::draw(const RectangleType & displayArea, const Point offset)
{
    bool result;
    WinSetUnderlineMode(solidUnderline);
    result = BLGlyph::draw(displayArea, offset);
    WinSetUnderlineMode(noUnderline);
    return result;
}

Point
WMLUnderline::getExtent(const RectangleType & displayArea, const Point offset)
{
    Point pt;
    WinSetUnderlineMode(solidUnderline);
    pt = BLGlyph::getExtent(displayArea, offset);
    WinSetUnderlineMode(noUnderline);
    return pt;
}

#pragma mark -----
WMLEmphasis::WMLEmphasis(BLGlyph * parent, XMLElement * elem)
: BLTextTags(parent, elem)
{
}

WMLEmphasis::~WMLEmphasis()
{
}

bool
WMLEmphasis::draw(const RectangleType & displayArea, const Point offset)
{
    FontID curFont;

```

```

bool result;
curFont = FntSetFont (boldFont);
result = BLGlyph::draw(displayArea, offset);
FntSetFont (curFont);
return result;

Point
WMLEmphasis::getExtent(const RectangleType & displayArea, const Point offset)
{
    FontID curFont;
    Point pt;
    curFont = FntSetFont (boldFont);
    pt = BLGlyph::getExtent(displayArea, offset);
    FntSetFont (curFont);
    return pt;
}

#pragma mark -----
WMLStrong::WMLStrong(BLGlyph * parent, XMLElement * elem)
{
    BLTextTags(parent, elem)
}

WMLStrong::~WMLStrong()
{
}

bool
WMLStrong::draw(const RectangleType & displayArea, const Point offset)
{
    FontID curFont;
    bool result;
    curFont = FntSetFont (largeBoldFont);
    result = BLGlyph::draw(displayArea, offset);
    FntSetFont (curFont);
    return result;
}

Pc
WML
{
    FontID curFont;
    Point pt;
    curFont = FntSetFont (largeBoldFont);
    pt = BLGlyph::getExtent(displayArea, offset);
    FntSetFont (curFont);
    return pt;
}

#pragma mark -----
WMLBig::WMLBig(BLGlyph * parent, XMLElement * elem)
{
    BLTextTags(parent, elem)
}

WMLBig::~WMLBig()
{
}

```

```

}

bool
WMLBig::draw(const RectangleType & displayArea, const Point offset)
{
    FontID curFont;
    bool result;
    curFont = FntSetFont (largeFont);
    result = BLGlyph::draw(displayArea, offset);
    FntSetFont (curFont);
    return result;
}

Point
WMLBig::getExtent(const RectangleType & displayArea, const Point offset)
{
    FontID curFont;
    Point pt;
    curFont = FntSetFont (largeFont);
    pt = BLGlyph::getExtent(displayArea, offset);
    FntSetFont (curFont);
    return pt;
}

// BLTextTags.h
#pragma once
#include "BLGlyph.h"
class BLTextTags : public BLGlyph
{
public:
    BLTextTags(BLGlyph * parent, XMLElement * elem);
    ~BLTextTags();
    virtual
        draw(const RectangleType & displayArea, const Point offset);
    virtual bool
        getExtent(const RectangleType & displayArea, const Point offset);
    static bool
        registerIfTextTag(BLGlyph * glyphP, XMLElement * child);
protected:
};

class WMLUnderline : public BLTextTags
{
public:
    WMLUnderline(BLGlyph * parent, XMLElement * elem);
    ~WMLUnderline();
    virtual
        draw(const RectangleType & displayArea, const Point offset);
    virtual bool
        getExtent(const RectangleType & displayArea, const Point offset);
protected:
};

class WMLEmphasis : public BLTextTags
{
public:
    WMLEmphasis(BLGlyph * parent, XMLElement * elem);
    ~WMLEmphasis();
    virtual
        draw(const RectangleType & displayArea, const Point offset);
    virtual bool
        getExtent(const RectangleType & displayArea, const Point offset);
};

```

# Source Code

Sword xj

```

protected:
};

class WMLStrong : public BLTextTags
{
public:
    WMLStrong(BLGlyph * parent, XMLElement * elem);
    ~WMLStrong();

    virtual
    draw(const RectangleType & displayArea, const Point offset);
    virtual bool
    getExtent(const RectangleType & displayArea, const Point offset);
    virtual Point

protected:
};

class WMLBig : public BLTextTags
{
public:
    WMLBig(BLGlyph * parent, XMLElement * elem);
    ~WMLBig();

    virtual
    draw(const RectangleType & displayArea, const Point offset);
    virtual bool
    getExtent(const RectangleType & displayArea, const Point offset);
    virtual Point

protected:
}; // BLTypes.h

#pragma once
class Point
{
public:
    Point() { x = 0; y = 0; }
    Point(const PointType &pt) { x = pt.x; y = pt.y; }
    Point(Sword x, Sword y) { x = x; y = y; }
    Point(const Point &p) { x = p.x; y = p.y; }

    PointType
    palmPoint() const
    {
        PointType p; p.x = x; p.y = y; return p;
    }

    Boolean
    isInside(const RectangleType &rect)
    {
        return RectPtInRectangle(x, y, (RectangleType &rect));
    }

    bool
    operator==(const Point &p) const { return (x == p.x) && (y == p.y); }
    bool
    operator!=(const Point &p) const { return !operator==(p); }

    Point
    operator-(const Point &p) { Point n;
        n.x = x-p.x;
        n.y = y-p.y;
        return n; }

    operator PointType() const { return palmPoint(); }

    void
    offset(Point p)
    {
        x += p.x;
        y += p.y;
    }

    static Point minPoint(const Point &p1, const Point &p2)
    {
        Point min;
        min.x = min(p1.x, p2.x);
        min.y = min(p1.y, p2.y);
        return min;
    }
};

class DataReader
{
public:
    RawPageData(bool unlock, bool deallocate, bool release, VoidHnd handle)
    {
        mUnlock = unlock;
        mDeallocate = deallocate;
        mRelease = release;
        mHandle = handle;
        mData = NULL;
        mLength = 0;
        mHandle = NULL;
    }
    ~RawPageData()
    {
        if(mUnlock) { MemHandleUnlock(mHandle); }
        if(mRelease) { DmReleaseResource(mHandle); }
        if(mDeallocate) delete[] mData;
    }

    bool isData()
    {
        return (mData != NULL) && (mLength > 0);
    }

    char *
    mData;
    UInt
    mLength;
    VoidHnd
    mHandle;

protected:
    bool
    mUnlock;
    bool
    mDeallocate;
    bool
    mRelease;
};

class DataReader
{

```

```

public:
    virtual ULONG readData(CharPtr buffer, ULONG length) = 0;
    virtual void readFailure() {}

class Timer
{
public:
    virtual bool checkTime() = 0;
};

// BLURLHandler.h
#pragma once
class BLURLHandler
{
public:
    virtual void goto(char * url, bool relativeURL) = 0;
    virtual void mailto(char * url) = 0;
};

// BLUtils.cpp
#include "BLUtils.h"
#include "BLVector.h"
#include "BLIterator.h"

CharPtr
BLUtils::cloneString(CharPtr str, UInt extraSpace)
{
    if (str == NULL)
        return NULL;

    CharPtr copy = new Char[StrLen(str) + 1 + extraSpace];

    if (copy != NULL)
        return StrCopy(copy, str);
    else
        throw OutOfMemoryException();

    return NULL;
}

bool
BLUtils::copyStringIfRoom(CharPtr dest, CharPtr source, int destSize)
{
    if (StrLen(source) > destSize)
        return false;

    StrCopy(dest, source);

    return true;
}

VoidPtr
BLUtils::getObjectPtr(Word objectId)
{
    FormPtr frmP;

    frmP = FrmGetActiveForm();
    return FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, objectId));
}

bool
BLUtils::isWhitespace(CharPtr text, int length)
{
    for (int i=0; i<length; i++)
    {
        if (!isWhitespace(text[i]))
            return false;
    }
}

```

```

bool
BLUtils::isWhitespace(Char c)
{
    return (c == ' ') || (c == '\n') || (c == '\r');
}

bool
BLUtils::isDigit(Char c)
{
    return (c >= '0') && (c <= '9');
}

bool
BLUtils::isHexDigit(Char c)
{
    return ((c >= 'a') && (c <= 'f')) ||
           ((c >= 'A') && (c <= 'F')) ||
           isDigit(c);
}

int
BLUtils::getNumericalValue(Char c)
{
    if (isDigit(c))
        return c - '0';
    else if ((c >= 'a') && (c <= 'f'))
        return 10 + c - 'a';
    else if ((c >= 'A') && (c <= 'F'))
        return 10 + c - 'A';
    else
        return -1;
}

int
BLUtils::hexToInt(CharPtr hexString, int bytesUsed, int bytesAvailable)
{
    int result = 0;
    int pos = 0;

    while (isHexDigit(hexString[pos]))
    {
        int value = getNumericalValue(hexString[pos]);

        result <<= 4;
        result |= value;
        pos++;

        if (pos == bytesAvailable)
            return -1;

        bytesUsed += pos;
        return result;
    }

    /*****
    * FUNCTION: getFocusObjectPtr
    * DESCRIPTION: This routine returns a pointer to the field object, in
    *               the current form, that has the focus.
    *****/
}

```

```

* PARAMETERS: nothing
* RETURNED: pointer to a field object or NULL if there is no focus
* .....
FieldPtr
BLUtils::getFieldFromHandle(fieldID, newHandle);
}

* .....
* FUNCTION: setFieldTextFromHandle
* DESCRIPTION: This routine sets the text handle given field id,
* PARAMETERS: fieldID
*             txth -- handle of the string
* RETURNED:
* .....
FieldPtr
BLUtils::setFieldTextFromHandle(Word fieldID, Handle txth)
{
    Handle oldTxth;
    FrmPtr frmP = FrmGetActiveForm();
    FieldPtr fldP;

    // get the field and the field's current text handle.
    fldP = (FieldPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, fieldID));
    ErrNonFatalDisplayIf(!fldP, "Missing field");
    oldTxth = FldGetTextHandle(fldP);

    // set the field's text to the new text.
    FldSetTextHandle(fldP, txth);
    //FldDrawField(fldP);

    // free the handle AFTER we call FldSetTextHandle().
    if (oldTxth)
        MemHandleFree(oldTxth);

    return fldP;
}

void
BLUtils::initBasicDialogForm(FrmPtr* frmP, Word frmID, FrmEventHandlerPtr handlerP, Word
focusFieldID)
{
    MenuEraseStatus(0);
    frmP = FrmInitForm(frmID);

    // Place cursor in field if not NULL
    if (focusFieldID)
        FrmSetFocus(frmP, FrmGetObjectIndex(frmP, focusFieldID));

    FrmSetActiveForm(frmP);
    if(handlerP)
        FrmSetEventHandler(frmP, handlerP);
}

bool
BLUtils::isInsideRectVector(Point p, BLVector<RectangleType, 4> & vector)
{
    BLIterator<RectangleType, 4> iter(vector);
    while(iter.hasNext())
    {
        RectangleType rect = iter.getNext();
        if(p.isInside(rect))
            return true;
    }
    return false;
}

```

```

* PARAMETERS: nothing
* RETURNED: pointer to a field object or NULL if there is no focus
* .....
FieldPtr
BLUtils::getFocusObjectPtr(void)
{
    FrmPtr frmP;
    Word focus;

    // Get a pointer to the active form and the index of the form object with focus.
    frmP = FrmGetActiveForm();
    focus = FrmGetFocus(frmP);

    // If no object has the focus return NULL pointer.
    if (focus == noFocus)
        return (NULL);

    // Return a pointer to the object with focus.
    return ((FieldPtr)FrmGetObjectPtr(frmP, focus));
}

* .....
* FUNCTION: setFieldToStrResource
* DESCRIPTION:
* PARAMETERS: fieldID
* RETURNED:
* .....
void
BLUtils::setFieldToStrResource(Word fieldID, Word strResID)
{
    CharPtr defaultProtocolStr;
    VoidHandle rchHandle;

    rchHandle = DeGetResource('tSTR', (Int) strResID);
    defaultProtocolStr = (CharPtr) MemHandleLock(rchHandle);
    BLUtils::writeStrToField(fieldID, defaultProtocolStr);
    MemHandleUnlock(rchHandle);
    DeReleaseResource(rchHandle);
}

* .....
* FUNCTION: writeStrToField
* DESCRIPTION: This routine allocates a handle and sets the field to the str
* PARAMETERS: fieldID
*             str -- string to write to the field
* RETURNED:
* .....
void
BLUtils::writeStrToField(Word fieldID, CharPtr str)
{
    Handle newHandle;

    // Allocate a new memory chunk that will contain a copy of the str.
    newHandle = (Handle)MemHandleNew(StrLen(str) + 1);

    // Copy the str to the locked handle.
    StrCopy((CharPtr)MemHandleLock(newHandle), str);

    // Unlock the new memory chunk.
    MemHandleUnlock(newHandle);
}

```

```

    }

    CharPtr
    BUUtils::IPtoString(ULONG ipAddress)
    {
        CharPtr string = new Char(4*4);
        int pos = 0;

        for(int i=0; i<4; i++)
        {
            StrToA(string + pos, (Long) ((ipAddress & 0xFF000000) >> 24));
            pos = (int) StrLen(string);
            string[pos++] = '.';
            ipAddress <<= 8;
        }

        string[pos - 1] = '\0';
        return string;
    }

    /* a bit is 1 if the corresponding char needs to be escaped */
#define BIT_PACK(b0, b1, b2, b3, b4, b5, b6, b7) (b7 << 7) | \
    (b6 << 6) | \
    (b5 << 5) | \
    (b4 << 4) | \
    (b3 << 3) | \
    (b2 << 2) | \
    (b1 << 1) | \
    (b0 << 0)

    const Byte kEscapeSymbol[0x80 / 8] =
    {
        /* 0x0 */ BIT_PACK(1, 1, 1, 1, 1, 1, 1, 1),
        /* 0x8 */ BIT_PACK(1, 1, 1, 1, 1, 1, 1, 1),
        /* 0x10 */ BIT_PACK(1, 1, 1, 1, 1, 1, 1, 1),
        /* 0x18 */ BIT_PACK(1, 1, 1, 1, 1, 1, 1, 1),
        /* 0x20 */ BIT_PACK(1, 1, 1, 1, 1, 1, 1, 1),
        /* 0x28 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x30 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x38 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x40 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x48 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x50 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x58 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x60 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x68 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x70 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0),
        /* 0x78 */ BIT_PACK(0, 0, 0, 0, 0, 0, 0, 0);
    };

    inline static bool
    checkBit(int num)
    {
        return (kEscapeSymbol[num / 8] & (1 << (num % 8))) != 0;
    }

    bool
    BUUtils::escapeChars(CharPtr dest, CharPtr source, UInt destSize)
    {
        // !!! Do rest of www.ietf.org/rfc/rfc2396.txt

        int sourcePos = 0;
        int destPos = 0;

        if (StrNCompare(source, "http://", StrLen("http://")) == 0)
        {
            StrCopy(dest, "http://");
        }
    }

    while(1)
    {
        if((destPos + 1) >= destSize)
            return false;

        bool escape = false;

        Byte c = (Byte) source[sourcePos];

        if(c == 0)
            break;
        else if(c > 0x80)
            escape = true;
        else
            escape = checkBit(c);

        if(escape)
        {
            Char sequence[16];
            CharPtr seqp = sequence;

            StrToH(sequence, (ULONG) c);

            while(*seqp == '0' && seqp)
                seqp++;

            StrToLower(seqp, seqp);

            seqp--;

            seqp[0] = '\\';

            if(!copyStringIfRoom(dest + destPos, seqp, (int) (destSize - destPos)))
                return false;

            destPos += (int) StrLen(seqp);
        }
        else
        {
            dest[destPos++] = (Char) c;
        }
        sourcePos++;

        if(dest[destPos] == '\\0')
            return true;
    }

    ULONG
    BUUtils::dateToSeconds(const DateType & date)
    {
        ULONG days = DateToDays(date);

        return days * 241 * 601 + 601;
    }

    /*
    Build 224: Cresenda (Alex Klyce). kExpirationDate = 196, 6, 101;
    Build 225: Atlanta potential investors. kExpirationDate = 196, 6, 101;
    Build 226: Cresenda 2
    Build 227: Palm
    Build 228:
    Build 229: Excite
    Build 230: Atlanta Angels
    */
    bool

```



```

BLUtils::checkExpirationDate()
{
    ULONG seconds = TimGetSeconds();
    ULONG expireSeconds = dateToSeconds(kExpirationDate);

    if(seconds < expireSeconds)
        return true;
    else
    {
        FrmCustomAlert(CustomAlert, "Build 230 has expired. Contact jkleid@blueark.com for
        up-to-date version.", "", "");
        return false;
    }
}

// BLUtils.h
#pragma once
#include "BLVectorHeader.h"
#include "BLTypes.h"

// BLUtils
{
    public:
        static CharPtr cloneString(CharPtr str, UInt extraSpace = 0);
        static bool copyStringIfRoom(CharPtr dest, CharPtr source, int destSize);
        static voidPtr getObjectPtr(Word objectID);
        static FieldPtr getFocusObjectPtr(void);
        static void setFieldToTrResource(Word fieldID, Word strHeadID);
        static void writeStrToField(Word fieldID, CharPtr str);
        static FieldPtr setFieldTextFromHandle(Word fieldID, Handle txtH);
        static void initBasicDialogForm(FormPtr frmP,
            Word frmID,
            FormEventHandlerPtr handlerP,
            Word focusFieldID);
        static bool isWhiteSpace(CharPtr text, int length);
        static bool isWhiteSpace(Char c);
        static bool isDigit(Char c);
        static bool isHexDigit(Char c);
        static int getNumericalValue(Char c);
        static int hexToInt(CharPtr hexString, int bytesUsed, int bytesAvailable);
        static bool isInsideRectVector(Point p, BLVector<RectangleType, 4> & vector);
        static CharPtr lPToString(ULONG lpAddress);
        static int abs(int i) { return i > 0 ? i : -i; }
        static long abs(long i) { return i > 0 ? i : -i; }
        static bool checkExpirationDate();
        static ULONG dateToSeconds(const DateType & date);
        static bool escapeChars(CharPtr dest, CharPtr source, UInt destSize);
};

class OutOfMemoryException
{
    // BLVector.h
#pragma once
#include <string.h>
#include "BLMemoryManager.h"
#include "BLVectorHeader.h"

```

```

template <class T, int increment_>
BLVectorManager<T, increment_>::BLVectorManager()
{
    BLVector<T, increment_>::mMemManager = NULL;
}

template <class T, int increment_>
BLVector<T, increment_>::BLVector(int initSize)
{
    if(initSize <= 0)
    {
        mPhysSize = 0;
        mLogSize = 0;
        mElements = NULL;
        return;
    }
    mPhysSize = initSize;
    mLogSize = 0;
    if(!mMemManager)
        mElements = new T[mPhysSize];
    else
        mElements = (T*) mMemManager->newChunk(sizeof(T) * mPhysSize);
}

template <class T, int increment_>
BLVector<T, increment_>::BLVector(const BLVector &original)
{
    mPhysSize = original.mPhysSize;
    mLogSize = original.mLogSize;
    if(mPhysSize == 0)
    {
        mElements = NULL;
        return;
    }
    if(!mMemManager)
        mElements = new T[mPhysSize];
    else
        mElements = (T*) mMemManager->newChunk(sizeof(T) * mPhysSize);
    memcpy(mElements, original.mElements, sizeof(T) * mLogSize);
}

template <class T, int increment_>
BLVector<T, increment_>::BLVector()
{
    if(!mMemManager)
        delete[] mElements;
}

template <class T, int increment_>
void
BLVector<T, increment_>::add(T item)
{
    expandIfNeeded(mLogSize);
    mElements[mLogSize++] = item;
}

template <class T, int increment_>
void
BLVector<T, increment_>::insert(T item, int pos)
{
    expandIfNeeded(mLogSize);
    MemMove(&mElements[pos+1], &mElements[pos], sizeof(T)*(mLogSize-pos));
    mElements[pos] = item;
    ++mLogSize;
}

```

```

class BLVectorManager;
template <class T, int increment_>
class BLVector
{
public:
    BLVector(int initSize = increment_);
    BLVector(const BLVector &original);
    ~BLVector();

    void add(T item);
    void insert(T item, int pos);
    void replace(T item, int pos) { mElements[pos] = item; }
    void remove(int pos);
    void removeAll() { mLogSize = 0; }
    void removeAll() { if(mLogSize > 0) mLogSize--; return }
    void removeLast() { if(mLogSize > 0) mLogSize--; return }
    void deleteAll();

    T& operator[](int index) { return mElements[index]; }
    const T& getElementAt(int index) const;
    T* getAllElements() const { return mElements; }
    int getNumElements() const { return mLogSize; }
    T getLastElement() const { if(mLogSize <= 0) return mElements[0]; return mElements[mLogSize - 1]; }
    T popLastElement() { if(mLogSize > 0) mLogSize--; return mElements[mLogSize]; }
    static void setMemManager(BLMemoryManager * memMan) { sMemManager = memMan; }

protected:
    static BLMemoryManager * sMemManager;
    void expandIfNeeded(int newLogSize);
    T* mElements;
    int mPhysSize;
    int mLogSize;
}; // BLView.cpp
#include "BLView.h"

BLView::BLView()
{
}

BLView::~BLView()
{
}

void BLView::erase()
{
    RectangleType bounds = getBounds();
    WinEraseRectangle(&bounds, 0);
} // BLView.h

#pragma once
#include "BLEventHandler.h"
class BLView : public BLEventHandler
{
public:
    BLView();
    ~BLView();
    virtual void open() = 0;

```

File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM

```

virtual bool doesHandleType(EventClasses event)
{ return event == KUIType; }

virtual Boolean handleEvent(EventPtr eventP) = 0;

virtual RectangleType getBounds() = 0;

virtual void draw() = 0;

virtual void erase();

// BLWMLActions.cpp
#include "BLWMLActions.h"
#include "BLWMLSession.h"
#include "BLUtils.h"

const CharPtr KURLenCode = "application/x-www-form-urlencoded";

const int kWepTicksPerSecond = 10;

cor int KURLBufferLen = 256;

WM .WMLGo()
{
    mHref = NULL;
    mSendReferer = false;
    mMethod = kGet;
    mEncodingType = KURLenCode;
    mAcceptCharset = NULL;
}

WMLGo::WMLGo(BLGlyph * parent, const WMLGo &original)
: WMLAction(parent)
{
    mHref = original.mHref;
    mSendReferer = original.mSendReferer;
    mMethod = original.mMethod;
    mEncodingType = original.mEncodingType;
    mAcceptCharset = original.mAcceptCharset;
}

void WMLGo::activate()
{
    WMLHyperlink link(mHref);
    nk.activate();
}

void WMLPrevious::activate()
{
    BLGlyph::getSession()->goBack();
}

void WMLRefresh::activate()
{
    BLGlyph::getSession()->refresh();
}

WMLHyperlink::WMLHyperlink(const WMLHyperlink &link)
{
    mHref = link.mHref;
}

void WMLHyperlink::activate()

```

```

Char urlBuffer(KURLBufferLen);
Char urlBuffer2(KURLBufferLen);

CharPtr tempBuffer = urlBuffer;
CharPtr escapedString = NULL;

UInt bufLength = KURLBufferLen;
UInt escapeBufLength = 0;

WMLVariables * vars = BLGlyph::getSession()->getVars();

if(!vars->insertVariables(tempBuffer, mHref, KURLBufferLen))
{
    bufLength = KURLBufferLen * 5;
    tempBuffer = new Char(bufLength);
    if(!tempBuffer)
        return;
    if(!vars->insertVariables(tempBuffer, mHref, bufLength))
        goto QUIT;
    escapeBufLength = (StrLen(tempBuffer) * 3) / 2;
    escapedString = new Char(escapeBufLength);
}
else
{
    escapedString = urlBuffer2;
    escapeBufLength = KURLBufferLen;
    tempBuffer = urlBuffer;
}

if(!BLUtils::escapeChars(escapedString, tempBuffer, escapeBufLength))
{
    if(escapedString != urlBuffer2)
        delete [] escapedString;
    escapeBufLength *= 2;
    escapedString = new Char(escapeBufLength);
    if(!escapedString)
        goto QUIT;
    if(!BLUtils::escapeChars(escapedString, tempBuffer, escapeBufLength))
    {
        goto QUIT;
    }
}

if(StrNCaslessCompare(escapedString, "mailto:", StrLen("mailto:")) == 0)
    BLGlyph::getSession()->mailto(escapedString);
else
    BLGlyph::getSession()->goTo(escapedString, true);

QUIT:

if(tempBuffer != urlBuffer)
    delete [] tempBuffer;

if(escapedString != urlBuffer2)
    delete [] escapedString;
}

WMLDo::WMLDo()
{
    mType = unknown;
    mName = NULL;
    mLabel = NULL;
    mIsOptional = false;
}

```

```

1 0 0 3 4 9 0 0
2 0 0 3 4 9 0 0
3 0 0 3 4 9 0 0
4 0 0 3 4 9 0 0
5 0 0 3 4 9 0 0
6 0 0 3 4 9 0 0
7 0 0 3 4 9 0 0
8 0 0 3 4 9 0 0
9 0 0 3 4 9 0 0
10 0 0 3 4 9 0 0

```

```

WMLDo::WMLDo(BLGlyph * parent, const WMLDo &original)
: WMLAction(parent)
{
    mType = original.mType;
    mName = original.mName;
    mLabel = original.mLabel;
    mIsOptional = original.mIsOptional;
}

WMLAnchor::WMLAnchor()
{
    mTitle = NULL;
    mAccessKey = NULL;
    mAction = NULL;
}

WMLAnchor::WMLAnchor(BLGlyph * parent, const WMLAnchor & original)
: WMLAction(parent)
{
    mTitle = original.mTitle;
    mAccessKey = original.mAccessKey;
    mAction = original.mAction;
}

WMLTimer::WMLTimer(WMLCard * parent, ULONG value)
: BLGlyph(parent)
{
    mEnclosingCard = parent;
    mValue = value;
    mTickValue = (SysTicksPerSecond() / kWapTicksPerSecond) * mValue;
    mOriginalTicks = 0;
    mIsTicking = false;
    parent->setTimer(this);
}

void
WMLTimer::start()
{
    mOriginalTicks = TimGetTicks();
    mIsTicking = true;
    BLGlyph::getSession()->setTimer(mTickValue, this);
}

WMLTimer::stop()
{
    mIsTicking = false;
}

bool
WMLTimer::checkTime()
{
    ULONG currentTime = TimGetTicks();
    if(mIsTicking && ((mOriginalTicks + mTickValue) <= currentTime))
    {
        WMLAction * action = mEnclosingCard->getIntrinsicEvents().mOnTimer;
        if(action != NULL)
            action->activate();
        return true;
    }
    return false;
}

WMLSetVar::WMLSetVar()
{
    mName = NULL;
    mValue = NULL;
}

WMLSetVar::WMLSetVar(BLGlyph * parent, const WMLSetVar & original)
: WMLAction(parent)
{
    mName = original.mName;
    mValue = original.mValue;
}

void
WMLSetVar::activate()
{
    WMLVariables * vars = getSession()->getVars();
    vars->add(mName, mValue);
}

WMLPostfield::WMLPostfield()
{
    mName = NULL;
    mValue = NULL;
}

WMLPostfield::WMLPostfield(BLGlyph * parent, const WMLPostfields original)
: WMLAction(parent)
{
    mName = original.mName;
    mValue = original.mValue;
}

void
WMLPostfield::activate()
{
    // BLWMLActions.h
    #pragma once
    #include "BLGlyph.h"

    class WMLAction : public BLGlyph
    {
    public:
        WMLAction(BLGlyph * parent = NULL) : BLGlyph(parent) {}

        virtual void activate() = 0;
        virtual void setAction(WMLAction * /*action*/) {}
        virtual WMLAction * getAction() { return NULL; }
    };

    class WMLPrevious : public WMLAction
    {
    public:
        WMLPrevious(BLGlyph * parent) : WMLAction(parent) {}

        virtual void activate();
    };
}

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

```

class WMLRefresh: public WMLAction
{
public:
    WMLRefresh(BLGlyph * parent) : WMLAction(parent) {}

    virtual void activate();

};

class WMLHyperlink : public WMLAction
{
public:
    WMLHyperlink(CharPtr url = NULL) { mHref = url; }
    WMLHyperlink(const WMLHyperlink &link);

    virtual void activate();

    CharPtr mHref;
};

class WMLGo : public WMLAction
{
public:
    enum HTTPMethod { kGet, kPost};

    WMLGo();
    WMLGo(BLGlyph * parent, const WMLGo &original);

    virtual void activate();

    friend class WBXMLParser;

protected:
    CharPtr mHref;
    bool mSendReferer;
    HTTPMethod mHTTPMethod;
    CharPtr mEncodingType;
    CharPtr mAcceptCharset; // !!! deal with this later
};

class WMLDo : public WMLAction
{
public:
    enum DoType { kUnknown = 0, kAccept, kPrev, kHelp, kReset, kOptions, kDelete };

    WMLDo();
    WMLDo(BLGlyph * parent, const WMLDo &original);

    virtual void activate()
    { if(mAction) mAction->activate(); }

    virtual void setAction(WMLAction * action)
    { mAction = action; }

    virtual WMLAction * getAction() { return mAction; }

    friend class WBXMLParser;
    friend class WMLCard;

protected:
    WMLAction * mAction;

    DoType mType;
    CharPtr mName;
    CharPtr mLabel;
    bool mIsOptional;
};

class WMLOnEvent : public WMLAction
{
public:
    WMLOnEvent(BLGlyph * parent) : WMLAction(parent) {}

    activate()
    { if(mAction) mAction->activate(); }

    virtual void setAction(WMLAction * action)
    { mAction = action; }

    virtual WMLAction * getAction() { return mAction; }

protected:
    WMLAction * mAction;
};

class WMLAnchor : public WMLAction
{
public:
    WMLAnchor();
    WMLAnchor(BLGlyph * parent, const WMLAnchor & original);

    virtual void activate()
    { if(mAction) mAction->activate(); }

    virtual void setAction(WMLAction * action)
    { mAction = action; }

    virtual WMLAction * getAction() { return mAction; }

    virtual BLVector<BLGlyph*, kChildrenInc> * children()
    { return mChildren; }

    friend class WBXMLParser;

protected:
    BLVector<BLGlyph*, kChildrenInc> mChildren;

    WMLAction * mAction;
    CharPtr mTitle;
    CharPtr mAccessKey;
};

class WMLTimer : public BLGlyph, public Timer
{
public:
    WMLTimer(WMLCard * parent, ULONG value);

    virtual void start();
    virtual void stop();
    virtual bool checkTime();

protected:
    WMLCard * mEnclosingCard;
    ULONG mValue;
    ULONG mTickValue;
    ULONG mOriginalTicks;
    bool mIsTicking;
};

```

```

class WMLHead : public BLGlyph
{
public:
    WMLHead(BLGlyph * parent) : BLGlyph(parent) {}
};

class WMLTemplate : public BLGlyph
{
public:
    WMLTemplate(BLGlyph * parent) : BLGlyph(parent), mEvents() {}

    WMLIntrinsicEvents &getIntrinsicEvents() { return mEvents; }

    friend class WBXMLParser;

protected:
    WMLIntrinsicEvents mEvents;

    class WMLSetvar : public WMLAction
    {
    public:
        WMLSetvar();
        WMLSetvar(BLGlyph * parent, const WMLSetvar& original);

        virtual void activate();

        friend class WBXMLParser;

    protected:
        CharPtr mName;
        CharPtr mValue;
    };

    class WMLPostfield : public WMLAction
    {
    public:
        WMLPostfield();
        WMLPostfield(BLGlyph * parent, const WMLPostfield& original);

        virtual void activate();

        friend class WBXMLParser;

    protected:
        CharPtr mName;
        CharPtr mValue;
    };

    // BLXMLForm.cpp
    #include "BLXMLForm.h"
    #include "BLXMLView.h"
    #include "BLIterator.h"
    #include "BLPrefs.h"
    #include "BLNetworking.h"
    #include <ScrDriver.h>

    BLXMLForm:BLXMLForm(
        UInt supportedFormID,
        UInt viewGadgetID,
        UInt vScrollBarID,
        UInt hScrollBarID)
    : BLForm(supportedFormID)
{
    mSupportedFormID = supportedFormID;
    mViewGadgetID = viewGadgetID;
    mVScrollBarID = vScrollBarID;
    mHScrollBarID = hScrollBarID;

    mbookmarkP = new BLBookmark();
    mWMLSession = NULL;
    mNetworkingP = networkingP;
    mIsOpened = false;

    BLXMLForm::~BLXMLForm()
    {
    }

    void
    BLXMLForm::formLoad(FormPtr formP)
    {
        BLForm::formLoad(formP);

        RectangleType rect;
        mWMLSession = new BLWMLSession(mNetworkingP);

        DWord depth = 2;

        if (mNetworkingP->getNetLibType() != BLNetworking::kNetLib)
            ScrDisplayMode(scrDisplayModeSet, NULL, NULL, &depth, NULL);

        FrmGetObjectBounds(mFormPtr,
            FrmGetObjectIndex(mFormPtr, mViewGadgetID),
            &rect);

        BLXMLView * wmlView = new BLXMLView(this, mWMLSession, rect, mVScrollBarID,
            mHScrollBarID);
        mWMLSession->setView(wmlView);
        addView(wmlView);

        mFocus = wmlView;

        mbookmarkP->updateBookmarkPopup(MainBookmarksPopListList);
    }

    void
    BLXMLForm::formOpen()
    {
        mIsOpened = true;

        FrmSetTitle(mFormPtr, BLUtils::cloneString("Blazer"));

        BLForm::formOpen();

        if (mFocus)
            mFocus->open();
    }

    void
    BLXMLForm::formClose()
    {
        if (mIsOpened)
        {
            mIsOpened = false;
            delete[] FrmSetTitle(mFormPtr);
            BLIterator<BLView*, KViewsInc> iter(mViews);
            delete mWMLSession;
        }
    }
}

```

06/15/2000 11:00:18 PM

```

mMMLSession = NULL;
while(iter.hasNext())
{
    delete iter.getNext();
}

Boolean
BLMMLForm::handleEvent(EventPtr eventP)
{
    Boolean handled = BLForm::handleEvent(eventP);
    switch(eventP->eType)
    {
        case ctSelectEvent:
            BLMMLView * view = (BLMMLView *) mViews.getElementAt(0);
            switch(eventP->data.ctSelect.ctrlID)
            {
                case MainLeftArrowButton:
                    mMMLSession->goBack();
                    handled = true;
                    break;
                case MainRightArrowButton:
                    mMMLSession->goForward();
                    handled = true;
                    break;
                case MainHomeButton:
                    mMMLSession->goHome();
                    handled = true;
                    break;
                case MainOpenButton:
                    doGotoPageDialog();
                    handled = true;
                    break;
            }
            break;
        case menuEvent:
            handled = doCommand(eventP->data.menu.itemID);
            break;
        case popSelectEvent:
            handled = bookmarkPopUpHandleEvent(eventP);
            break;
    }
    return handled;
}

Boolean
BLMMLForm::doCommand(Word command)
{
    Boolean handled = false;
    FormPtr frmP;
    switch(command)
    {
        case BrowserPageGoto:
            doGotoPageDialog();
            if(!mFocus)
                mFocus->draw();
            handled = true;
            break;
        case BrowserEditBookmarks:
            doBookmarkEditDialog();
            if(!mFocus)
                mFocus->draw();
    }
}

```

```

// nav commands
case BrowserPageBack:
    mMMLSession->goBack();
    handled = true;
    break;
case BrowserPageForward:
    mMMLSession->goForward();
    handled = true;
    break;
case BrowserPageHome:
    mMMLSession->goHome();
    handled = true;
    break;
case BrowserPageStop:
    handled = true;
    break;
/*
case BrowserPageReload:
    handled = true;
    break;
*/
case OptionsPreferences:
    doPrefPageDialog();
    handled = true;
    break;
case OptionsAboutBlazer:
    MenuEraseStatus(0);
    frmP = FrmInitForm(BlazerAboutForm);
    FrmDoDialog(frmP);
    FrmDeleteForm(frmP);
    if(!mFocus)
        mFocus->draw();
    handled = true;
    break;
/*
// Text Selection
case BrowserEditCut:
    fldP = BLUtils::getFocusObjectPtr();
    if (fldP)
        fldCut(fldP);
    handled = true;
    break;
case BrowserEditPaste:
    fldP = BLUtils::getFocusObjectPtr();
    if (fldP)
        fldCopy(fldP);
    handled = true;
    break;
case BookmarkEditPaste:
    fldP = BLUtils::getFocusObjectPtr();
    if (fldP)
        fldPaste(fldP);
    handled = true;
    break;
case BrowserEditSelectAll:
    fldP = BLUtils::getFocusObjectPtr();
    if (fldP)
        fldSetSelection(fldP, 0, fldGetTextLength(fldP));
    handled = true;
    break;
*/
return handled;
}

```

10054530 10054530

```

Boolean
BLMWLForm::bookmarkPopupHandleEvent(EventPtr eventP)
{
    ListPtr listP;
    Word indexSelected, lastIndexSelected, numElements;
    Boolean handled = false;
    Word id = 0;
    BLMWLView* view = 0;

    listP = (ListPtr) eventP->data.popSelect.listP;
    numElements = listP->getNumberOfItems(listP);
    indexSelected = eventP->data.popSelect.selection;
    lastIndexSelected = eventP->data.popSelect.priorSelection;

    // go to edit bookmarks form if the user selected the last item
    if (indexSelected == (numElements - 1))
    {
        doBookmarksEditDialog();
        handled = true;
    }
    otherwise go to the associated url
    else
    {
        CharPtr url = mbookmarkP->getUrl(indexSelected);
        if (!mWLSession != NULL) && (url != NULL)
        {
            mWLSession->goTo(url, false);
            handled = true;
        }
        delete[] url;
    }
    return handled;
}

void
BLMWLForm::doGotoPageDialog(void)
{
    FormPtr previousForm, frmP;

    previousForm = FrmGetActiveForm();
    BLUtils::initBasicDialogForm(frmP, GotoPageForm, gotoPageWrapper, GotoPageURLField);
    BLUtils::initBasicDialogForm(frmP, GotoPageForm, gotoPageWrapper, GotoPageURLField);
    CharPtr lastUrl;
    if (lastUrl = mbookmarkP->getLastOpenUrlP()) != NULL
    {
        BLUtils::setFieldToStrResource(GotoPageURLField, DefaultProtocolString);
        lastUrl = StrLen("http://");
        BLUtils::writeStrToField(GotoPageURLField, lastUrl);
        UInt startHilite = 0;
        UInt endHilite = StrLen(lastUrl);
        if (StrCaseLessCompare(lastUrl, "http://", StrLen("http://")) == 0)
            startHilite = StrLen("http://");
        FldSetSelection((FldPtr) FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP, GotoPageURLField)), startHilite, endHilite);
    }
    mbookmarkP->initGotoPageDialog();
    FrmDoDialog(frmP);
    CharPtr url = BLUtils::cloneString(mbookmarkP->getUrlOpenUrl());
    FrmDeleteForm(frmP);
    FrmSetActiveForm(previousForm);
    if (url)
        mWLSession->goTo(url, false);
    else if (!mFocus)
        mFocus->draw();
}

```

```

void
BLMWLForm::doBookmarksEditDialog(void)
{
    FormPtr previousForm, frmP;

    previousForm = FrmGetActiveForm();
    BLUtils::initBasicDialogForm(frmP, BookmarksEditForm, bookmarksEditWrapper, NULL);
    mbookmarkP->initBookmarkEditForm(BookmarksEditListList);
    FrmDoDialog(frmP);
    FrmDeleteForm(frmP);
    FrmSetActiveForm(previousForm);
    // update the bookmarks list for the popup
    mbookmarkP->updateBookmarksPopup(MainBookmarksPopListList);

    if (!mFocus)
        mFocus->draw();
}

void
BLMWLForm::doPrefsPageDialog(void)
{
    FormPtr previousForm, frmP;

    previousForm = FrmGetActiveForm();
    BLUtils::initBasicDialogForm(frmP, PreferencesForm, NULL, 0);
    BLUtils::writeStrToField(PreferencesServerField, BLPrefs::sPrefs.mServerAddress);
    CtlSetValue((CtlPtr) BLUtils::getObjectPtr(PreferencesShowImagesCheckBox),
        BLPrefs::sPrefs.mShowImages);
    if (FrmDoDialog(frmP) == PreferencesOKButton)
    {
        CharPtr serverAddressP = FldGetTextPtr((FldPtr)
            BLUtils::getObjectPtr(PreferencesServerField));
        if (serverAddressP)
        {
            StrCopy(BLPrefs::sPrefs.mServerAddress, serverAddressP);
        }
        BLPrefs::sPrefs.mShowImages =
            (CtlGetValue((CtlPtr) BLUtils::getObjectPtr(PreferencesShowImagesCheckBox)) != 0);
        FrmDeleteForm(frmP);
        FrmSetActiveForm(previousForm);
    }
    Boolean
    BLMWLForm::gotoPageWrapper(EventPtr eventP)
    {
        return mbookmarkP->gotoPageHandleEvent(eventP);
    }
    Boolean
    BLMWLForm::bookmarksEditWrapper(EventPtr eventP)
    {
        return mbookmarkP->bookmarksEditHandleEvent(eventP);
    }
    Boolean
    BLMWLForm::prefPageHandler(EventPtr eventP)
    {
        Boolean handled = false;
        Boolean showImages = false;
        switch (eventP->eType)
        {
            case ctlSelectEvent:

```



```

File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM

    if (eventP->data.ctrlEnter.controlID == PreferencesOKButton)
    {
        // Get the current setting of the show images checkbox.
    }
    break;
}

return(handled);
}

// BLWMLForm.h
#include "BLForm.h"
#include "BLWMLSession.h"
#include "BLBookmark.h"
#include "BLNetworking.h"

static BLBookmark* mbookmarkP;

class BLWMLForm : public BLForm
{
public:
    BLWMLForm( BLNetworking * networking,
               UInt supportedFormID,
               UInt viewGadgetID,
               UInt vScrollbarID,
               UInt hScrollbarID);
    ~BLWMLForm();

    virtual void formLoad(FormPtr formP);
    virtual void formOpen();
    virtual void formClose();

    virtual Boolean handleEvent(EventPtr eventP);

protected:
    BLWMLSession * mWMLSession;

    bool misOpened;

    UInt mSupportedFormID;
    UInt mViewGadgetID;
    UInt mVScrollbarID;
    UInt mHScrollbarID;
    BLNetworking * mNetworkingP;

    Boolean bookmarkPopUpHandled;
    Boolean bookmarkPopUpHandleEvent(EventPtr eventP);

    void doGoToPageDialog(void);
    void doBookmarksEditDialog(void);
    void doPrefsPageDialog(void);

    static Boolean gotoPageWrapper(EventPtr eventP);
    static Boolean bookmarksEditWrapper(EventPtr eventP);
    static Boolean prefsPageHandler(EventPtr eventP);

}; // BLWMLForm.cpp

#include "BLWMLInput.h"
#include "BLWMLActions.h"
#include "BLGlyph.h"
#include "BLUtils.h"
#include "BLWMLSession.h"

const UInt kCheckBoxSize = 6;
const Int kListID = 2222;
const Int kTriggerID = 1111;
const Int kMinWidth = 10;

WMLFieldSet::WMLFieldSet()
{
    mTitle = NULL;
}

```

Page: 101

```

WMLFieldSet:WMLField
: BGLlyph(parent)
{
    mTitle = origName
}

WMLInput:WMLInput(
{
    mName = NULL
    mValue = NULL
    mAccessKey = NULL
    mTitle = NULL
    mFormat = NULL
    mSize = -1
    mMaxLength = -1
    mTabIndex = -1
    mType = kTextField
    mEmptytok = false
}

WMLInput:WMLInput
: WMLInputElement
{
    mName = orig
    mValue = orig
    mAccessKey = orig
    mTitle = orig
    mFormat = orig
    mSize = orig
    mMaxLength = orig
    mTabIndex = orig
    mType = orig
    mEmptytok = orig
    WMLVariables *
    mTextEntered =
    if (!mTextEntered)
    {
        mTextEntered = orig
        if (mTextEntered)
        {
            vars->
        }
    }
    void
    WMLInput::draw( c
    {
        RectangleType
        rect.topLeft
        rect.topLeft
        rect.topLeft
        rect.extent
        rect.extent.x
        rect.extent.y
        WinDrawGrayRay
        if (mTextEntered)
        {
            TextStat
            if (mType

```

```

    WinDrawTruncChars(mTextEntered, StrLen(mTextEntered), ++rect.topLeft.x, rect.topLeft.y,
    --rect.extent.y, --rect.extent.x);
    else if (mType == kPassword) && (mTextEntered[0] != '\0')
    {
        CharPtr starChars = "*****";
        WinDrawTruncChars(starChars, StrLen(starChars), ++rect.topLeft.x, rect.topLeft.y,
        --rect.extent.x);
    }
}

void
WMInput::computeMetrics(int lineWidth)
{
    mLineWidth = lineWidth;

    bool
    WMInput::click(const RectangleType & displayArea,
        const Point & offset,
        PointType p)
    {
        bool inRect = false;
        RectangleType rect = WMLineElement::getArea(displayArea, offset);
        Point pt(p);
        inRect = pt.isInside(rect);
        if (inRect && (BLGlyph::sHitTestAction == kFindSelection))
            BLGlyph::sClickedAction = this;
        else if (BLGlyph::sClickedAction == this)
        {
            BLGlyph::sActiveRectVector.add(rect);
            WinInvertRectangle(&rect, 0);
        }
        return inRect;
    }

    void
    WMInput::activate()
    {
        // show text entry window
        mState::enableBasicState();
        if (doInputDialog())
        {
            WMVariables * vars = getSession()->getVars();
            vars->add(mName, mTextEntered);
            BLGlyph::getSession()->refresh(true);
        }
    }

    Point
    WMInput::getExtent(const Point & areaExtent)
    {
        TextState::enableBasicState();
        Point extent;
        LineMetrics metrics = getMetrics();
        extent.x = metrics.width;
        extent.y = metrics.height;
        return extent;
    }
}

LineMetrics
WMInput::getMetrics()
{
    TextState::enableBasicState();
    LineMetrics metrics;
    metrics.leadingSpace = 0;
    metrics.width = mLineWidth;
    metrics.baselineOffset = FntBaseline() + 1;
    metrics.overhang = FntDescenderHeight() + 1;
    return metrics;
}

Boolean
WMInput::bookmarksAddHandleEvent(EventPtr /*eventPtr*/)
{
    Boolean handled = false;
    return handled;
}

bool
WMInput::doInputDialog()
{
    bool valueChanged = false;
    FrmPtr frmP, previousForm;
    previousForm = FrmGetActiveForm();
    BLUtils::initBasicDialogForm(frmP, TextEntryForm, bookmarksAddHandleEvent,
    TextEntryTextField);
    if (mTextEntered != NULL)
        BLUtils::writeStrToField(TextEntryTextField, mTextEntered);
    if (FrmDoDialog(frmP) == TextEntryOKButton)
    {
        CharPtr serverAddressP = FldGetTextPtr((FldPtr)
        BLUtils::getObjectPtr(TextEntryTextField));
        delete () mTextEntered;
        mTextEntered = BLUtils::cloneString(serverAddressP);
        valueChanged = true;
    }
    FrmDeleteForm(frmP);
    FrmSetActiveForm(previousForm);
    return valueChanged;
}

WMSelect::WMSelect()
{
    mTitle = NULL;
    mName = NULL;
    mValue = NULL;
    mName = NULL;
    mValue = NULL;
    mTabIndex = -1;
    mType = kRadioButton;
    mSelectedP = NULL;
    mPreviousSelectedP = NULL;
    mNumOptions = 0;
}

```

```

    if(mTitle)
        widthNeeded = FntCharWidth(mTitle, StrLen(mTitle));
    if(widthNeeded < kMinWidth)
        widthNeeded = kMinWidth;
    for(int i=0; i<mNumOptions; i++)
    {
        WMLOption * optionP = dynamic_cast<WMLOption *>(children()->getElementAt(i));
        if(optionP == NULL)
        {
            items[i] = NULL;
            continue;
        }
        items[i] = optionP->getTitle();
        int width = FntCharWidth(items[i], StrLen(items[i])) + 5;
        if(widthNeeded < width)
            widthNeeded = width;
        if(widthNeeded > mLineWidth)
            widthNeeded = mLineWidth;
        FormPtr activeP = FrmGetActiveForm();
        Err err = ListNewList(
            activeP,
            kListID, // list id,
            displayArea.topLeft.x,
            displayArea.topLeft.y,
            widthNeeded,
            displayArea.extent.y,
            stdFont,
            min(displayArea.extent.y / FntLineHeight(), mNumOptions),
            kTriggerID; // trigger id
        if(err)
            return true;
        ListPtr listP = (ListPtr) BLUtlis::getObjectPtr(kListID);
        ListSetListChoices(listP, items, mNumOptions);
        short selection = ListPopupList(listP);
        err = FrmRemoveObject(activeP, FrmGetObjectIndex(activeP, kListID));
        getSession()->getView()->refreshFormPointer();
        return true;
    }
    else
        return false;
}
return false;
}

void
WMLSelect::draw(const RectangleType & displayArea,
                const Point & offset)
{
    RectangleType rect = displayArea;
    rect.topLeft.x = displayArea.topLeft.x - offset.x;
    rect.topLeft.y = displayArea.topLeft.y - offset.y;
    if(mType == kPopupList)
    {
        void * handle = DmGetResource('Tbmp', DownArrowBitmap);
        BitmapPtr bitMapP = (BitmapPtr) MemHandleLock(handle);
    }
}

```

```

    if(mTitle)
        widthNeeded = FntCharWidth(mTitle, StrLen(mTitle));
    if(widthNeeded < kMinWidth)
        widthNeeded = kMinWidth;
    for(int i=0; i<mNumOptions; i++)
    {
        WMLOption * optionP = dynamic_cast<WMLOption *>(children()->getElementAt(i));
        if(optionP == NULL)
        {
            items[i] = NULL;
            continue;
        }
        items[i] = optionP->getTitle();
        int width = FntCharWidth(items[i], StrLen(items[i])) + 5;
        if(widthNeeded < width)
            widthNeeded = width;
        if(widthNeeded > mLineWidth)
            widthNeeded = mLineWidth;
        FormPtr activeP = FrmGetActiveForm();
        Err err = ListNewList(
            activeP,
            kListID, // list id,
            displayArea.topLeft.x,
            displayArea.topLeft.y,
            widthNeeded,
            displayArea.extent.y,
            stdFont,
            min(displayArea.extent.y / FntLineHeight(), mNumOptions),
            kTriggerID; // trigger id
        if(err)
            return true;
        ListPtr listP = (ListPtr) BLUtlis::getObjectPtr(kListID);
        ListSetListChoices(listP, items, mNumOptions);
        short selection = ListPopupList(listP);
        err = FrmRemoveObject(activeP, FrmGetObjectIndex(activeP, kListID));
        getSession()->getView()->refreshFormPointer();
        return true;
    }
    else
        return false;
}
return false;
}

void
WMLSelect::draw(const RectangleType & displayArea,
                const Point & offset)
{
    RectangleType rect = displayArea;
    rect.topLeft.x = displayArea.topLeft.x - offset.x;
    rect.topLeft.y = displayArea.topLeft.y - offset.y;
    if(mType == kPopupList)
    {
        void * handle = DmGetResource('Tbmp', DownArrowBitmap);
        BitmapPtr bitMapP = (BitmapPtr) MemHandleLock(handle);
    }
}

```

```

    WinDrawBitmap(bitmapP, rect.topLeft.x, rect.topLeft.y);

    UInt startPos = bitmapP->width;
    MemHandleUnlock(handle);
    DnReleaseResource(handle);

    CharPtr text;
    if (getSelected())
        text = getSelected()->getTitle();
    else
        text = getTitle();
    if (text != NULL)
    {
        TextState::enableBasicState();
        WinDrawChars( (ConstCharPtr) text, StrLen(text),
            rect.topLeft.x + startPos,
            rect.topLeft.y);
    }

    Point
    WMLSelect::getExtent(const Point &areaExtent)
    {
        TextState::enableBasicState();
        return Point(mLineWidth, mLineHeight());
    }

    LineMetrics
    WMLSelect::getMetrics()
    {
        LineMetrics metrics;
        Point extent = getExtent(Point::zeroPoint());
        metrics.width = extent.x;
        metrics.baselineOffset = extent.y;
        metrics.overhang = 0;
        metrics.leadingSpace = 0;
        return metrics;
    }

    void
    WMLSelect::activate()
    {
    }

    WMLOptgroup::WMLOptgroup()
    {
        mTitle = NULL;
    }

    WMLOptgroup::WMLOptgroup(BLGLYPH * parent, const WMLOptgroup& original)
    : BLGLYPH(parent)
    {
        mTitle = original.mTitle;
    }

    RectangleType
    WMLOption::isLastSelectionArea;
    WMLOption::WMLOption(WMLSelect * currSelectP)

```



```

doDraw = true;
}
else
{
    if(mSelectP->mType == WMSelect::kRadioButtons)
    {
        WMOption * lastOptionP = mSelectP->getPreviousSelected();
        if(lastOptionP && (lastOptionP != this))
        {
            lastOptionP->erase(sLastSelectionArea, Point::sZeroPoint());
            lastOptionP->draw(sLastSelectionArea, Point::sZeroPoint());
        }
        mSelectP->setPreviousSelected(NULL);
    }
}

if(doDraw)
{
    erase(displayArea, offset);
    draw(displayArea, offset);
}

return inRect;
}

void
WMOption::erase( const RectangleType & displayArea,
                 const Point &offset)
{
    RectangleType rect = displayArea;
    rect.topLeft.x -= offset.x - 1;
    rect.topLeft.y -= offset.y - 1;
    rect.extent.x = rect.extent.y = kCheckBoxSize + 1;
    WinEraseRectangle(&rect, 0);
}

void
WMOption::deselect()
{
    CharPtr
    WMOption::getTitle()
    {
        if(mTitle)
            return mTitle;
        else if(mValue)
            return mValue;
        else
            return "[Option]";
    }

    WMButton:WMButton( BLGlyph * parent,
                       CharPtr text,
                       int length,
                       TextState state)
    : WMLineElement(parent)
    {
        mText = text;
        mLength = length;
        mState = state;
    }
}

```

```

void
WMButton::draw(const RectangleType & displayArea,
               const Point &offset,
               bool invert)
{
    RectangleType rect = displayArea;
    TextState::enableBasicState();
    rect.topLeft.x -= offset.x - 1;
    rect.topLeft.y -= offset.y - 1;
    rect.extent = getExtent();
    rect.topLeft.y++;
    rect.extent.x -= 3;
    rect.extent.y -= 3;
    if(!invert)
        WinDrawRectangleFrame(roundFrame, &rect);
    else
        WinDrawRectangle(&rect, 4);
    if(!invert)
        WinDrawChars(mText, (Word) mLength, (SWord) rect.topLeft.x + 3, (SWord)
        rect.topLeft.y + 1);
    else
        WinDrawInvertedChars(mText, (Word) mLength, (SWord) rect.topLeft.x + 3, (SWord)
        rect.topLeft.y + 1);
}

bool
WMButton::click( const RectangleType & displayArea,
                  const Point &offset,
                  PointType p)
{
    RectangleType rect = WMLineElement::getArea(displayArea, offset);
    Point pt(p);
    bool inRect = pt.isInside(&rect);
    if(BLGlyph::sHighlightAction == kFindSelection)
    {
        if(inRect)
        {
            BLGlyph::sClickedAction = this;
            return true;
        }
        else
            return false;
    }
    else if(BLGlyph::sHighlightAction == kHighlight)
    {
        if(inRect)
        {
            BLGlyph::sActiveRectVector.add(&rect);
            draw(displayArea, offset, true);
            return true;
        }
        else return false;
    }
    else if(BLGlyph::sHighlightAction == kUnhighlight)
    {
        if(BLGlyph::sClickedAction == this)
        {
            rect.extent = getExtent();
            WinEraseRectangle(&rect, 0);
            draw(displayArea, offset, false);
        }
    }
}

```

protected:

```

    }
    return inRect;
}

Point
WMLButton::getExtent()
{
    Point extent;
    LineMetrics metrics = getMetrics();
    extent.x = metrics.width;
    extent.y = metrics.height();
    return extent;
}

LineMetrics
WMLButton::getMetrics()
{
    LineMetrics metrics;
    metrics.baselineOffset = FntBaseline() + 1;
    metrics.overhang = FntDescenderHeight() + 4;
    TextState state = TextState::getBasicState();
    metrics.width = WMLTextRun::getTextWidth(mText, mLength, state, metrics.height()) + 9;
    return metrics;
}

void
WMLButton::activate()
{
    if(mState.action)
        mState.action->activate();
}

// BLWMLInput.h
#pragma once
#include "BigGlyph.h"
#include "BLWMLActions.h"

class WMLInput : public WMMLineElement, public virtual WMLAction
{
public:
    enum InputType { kText, kPassword };

    virtual void draw( const RectangleType & displayArea,
        const Point &offset);
    virtual void computeMetrics(int lineWidth);
    virtual bool click( const RectangleType & displayArea,
        const Point &offset,
        PointType p);
    virtual Point getExtent(const Point &areaExtent);
    LineMetrics getMetrics();
    void activate();
};

```

```

virtual Point
virtual Point
    LineMetrics
        getMetrics();
        activate();

// !!! Later, add KScrollList to this: (mType == kPopupMenu); )
bool
    IsDisplayable() { return (mType == kPopupMenu); )

void
    IncrementOptionCount() { mNumOptions++; }

computeMetrics(int lineWidth);

friend class
friend class
    WBXMLParser;
    WMLOption;

protected:
    WMLOption * mSelectedP;
    WMLOption * mPreviousSelectedP;

    CharPtr
    mName;
    CharPtr
    mValue;
    CharPtr
    mName;
    CharPtr
    mValue;
    int
    mTabIndex;
    SelectType mType;

    int
    mNumOptions;
    int
    mLineWidth;

};

class WMLOptgroup : public BGLGlyph
{
public:
    WMLOptgroup(BGLGlyph * parent, const WMLOptgroup& original);

friend class
    WBXMLParser;

protected:
    CharPtr
    mTitle;

    WMLOption : public WMLLineElement,
        virtual public BLParentGlyph,
        virtual public WMLAction

    public:
        WMLOption(WMLSelect * currSelectP);
        WMLOption(BGLGlyph * parent, const WMLOption& original);

        virtual void
            draw( const RectangleType & displayArea,
                  const Point &offset);

        virtual bool
            click( const RectangleType & displayArea,
                   const Point &offset,
                   PointType p);

        void
            deselect();

        virtual Point
            getExtent(const Point &areaExtent);
        virtual Point
            getExtent() { return getExtent(Point::sZeroPoint()); }

        LineMetrics
            getMetrics();

        void
            activate();

```

```

        CharPtr
        mTitle;
        int
        StrLen(mTitle);

friend class
    WBXMLParser;

protected:
    void
        erase( const RectangleType & displayArea,
              const Point &offset);

    static
        RectangleType
        sLastSelectionArea;

    WMLSelect * mSelectP;

    CharPtr
    mValue;
    CharPtr
    mTitle;
    CharPtr
    mOnPick;

    bool
    misSelected;

};

class WMLButton : public WMLLineElement,
    virtual public WMLAction
{
public:
    WMLButton( BGLGlyph * parent,
               CharPtr text,
               int length,
               TextState state);

    virtual LineMetrics getMetrics();

    virtual void
        draw( const RectangleType & displayArea,
              const Point &offset)
            { draw(displayArea, offset, false); }

    virtual void
        draw( const RectangleType & displayArea,
              const Point &offset,
              bool invert);

    virtual bool
        click( const RectangleType & displayArea,
               const Point &offset,
               PointType p);

    virtual Point
        getExtent();

    void
        activate();

protected:
    CharPtr
    mText;
    int
    mLength;
    TextState
    mState;

};

// BLWMLSession.cpp
#include "BLWMLSession.h"
#include "BLUtils.h"
#include "BLPrefs.h"

const UInt kRenderIncrement = 2048;
const UInt kMaxPortStringSize = 8;
const int kDefaultPort = 80;
const int kChunkLengthBytes = 32;

extern Word AppNetRefnum;

```



```

BLWMLSession::BLWMLSession(BLNetworking * networkingP)
{
    mNetworkingP = networkingP;

    mCurrPage = -1;
    mLastPage = -1;
    mView = NULL;
    mDestCardID = NULL;
    mParseP = NULL;
    mIsDone = true;
    mPendingDownload = false;
    mDestURL = NULL;
    mContentLocation = NULL;
    mCurrImage = NULL;

    initializeSession();
    mCache.open();

    BLGlyph::setSession(this);
    mNavigationAction = kNone;

    BLWMLSession::~BLWMLSession()
    {
        mCache.closeRecords();
        mCache.close();
    }

    CharPtr
    BLWMLSession::convertRelativeURL(CharPtr baseURL, CharPtr relativeURL)
    {
        if(StrNCaslessCompare(relativeURL, "http://", StrLen("http://")) == 0)
        {
            return BLUtils::cloneString(relativeURL, 1);
        }

        int endPos = (int) StrLen(baseURL) - 1;
        int depth = 1;
        if(*relativeURL == '/')
        {
            relativeURL++;
            depth = 0;
        }
        do
        {
            while(1)
            {
                if((relativeURL[0] == '.') && (relativeURL[1] == '.') && (relativeURL[2] == '/'))
                {
                    relativeURL += 3;
                    depth++;
                }
                else if((relativeURL[0] == '.') && (relativeURL[1] == '.'))
                {
                    relativeURL += 2;
                    depth++;
                    break;
                }
                else
                {
                    break;
                }
            }
        }
        while(1)
        {
            if((relativeURL[0] == '.') && (relativeURL[1] == '.') && (relativeURL[2] == '/'))
            {
                relativeURL += 3;
                depth++;
            }
            else if((relativeURL[0] == '.') && (relativeURL[1] == '.'))
            {
                relativeURL += 2;
                depth++;
                break;
            }
            else
            {
                break;
            }
        }
        if((relativeURL[0] == '.') && (relativeURL[1] == '.'))
        {
            relativeURL += 2;
        }
    }

    void
    BLWMLSession::verifyHTTPSlash(CharPtr url)
    {
        UInt offset = 0;
        if(url[0] && (StrNCaslessCompare(url, "http://", StrLen("http://")) == 0))
        {
            offset = StrLen("http://");
        }
        if(StrStr(url + offset, "/") == NULL)
    }

    if(depth > 1)
    // correct for bad ../ in URLs:
    // !!! imperfect, when / is not directory separator:
    int numSlashes = 0;
    int basePos = 0;
    while(baseURL[basePos] && (numSlashes < 3))
    {
        if(baseURL[basePos] == '/')
            numSlashes++;
        basePos++;
    }
    if(numSlashes < (depth + 3))
        depth = numSlashes - 3;
    if(depth < 0)
        depth = 0;

    if(depth == 0)
    {
        endPos = (int) StrLen("http://");
        while(baseURL[endPos++] != '/')
        {
            if(baseURL[endPos] == '\0')
                break;
        }
    }
    else
    {
        if(baseURL[endPos] == '/')
        {
            if(depth > 1)
                endPos--;
            else
                depth = 0;
        }
        while(depth > 0)
        {
            do
            {
                endPos--;
            } while((endPos > 0) && (baseURL[endPos] != '/'));
            depth--;
            if(endPos == 0)
                break;
        }
    }
    endPos++;

    CharPtr newURL = new Char[endPos + StrLen(relativeURL) + 2];
    StrNCopy(newURL, baseURL, (UShort) endPos);
    StrCopy(newURL + endPos, relativeURL);
    return newURL;
}

void
BLWMLSession::verifyHTTPSlash(CharPtr url)
{
    UInt offset = 0;
    if(url[0] && (StrNCaslessCompare(url, "http://", StrLen("http://")) == 0))
    {
        offset = StrLen("http://");
    }
    if(StrStr(url + offset, "/") == NULL)

```

```

        StrCopy(url + StrLen(url), "/");
    }

    void
    BLWMLSession::goTo(CharPtr url, bool isRelativeURL)
    {
        if(StrLen(url) == 0)
            return;

        BLHistoryItem * item = NULL;
        CharPtr currPageURL = mContentLocation;
        if(currPageURL && currPageURL[0] == '\0')
            currPageURL = NULL;

        if((currPageURL == NULL) && (mCurrPage >= 0))
            currPageURL = mPageHistory.getElementAt(mCurrPage)->getURL();

        if(currPageURL == NULL)
            currPageURL = mNetworking->getContentLocation();

        if((url[0] == '#') && (url[1] != '\0'))
        {
            // we're going to another card on same page
            item = new BLHistoryItem( BLUtils::cloneString(currPageURL),
                                     BLUtils::cloneString(url[1]));
        }
        else
        {
            if(currPageURL && isRelativeURL)
                url = convertRelativeURL(currPageURL, url);
            else
                url = BLUtils::cloneString(url, 1);

            verifyHTTPSlash(url);
            CharPtr cardName = extractCardName(url);
            item = new BLHistoryItem(url, cardName);
        }

        if((mPageHistory.getNumElements() - 1) >= 0)
            for(int i = (mPageHistory.getNumElements() - 1); i >= (mCurrPage + 1); i--)
            {
                delete mPageHistory.remove(i);
                if(mLastPage > i)
                    mLastPage--;
                else if(mLastPage == i)
                    mLastPage = -1;
            }

        if(item != NULL)
        {
            addToHistory(item);
            loadPage(item);
        }

        void
        BLWMLSession::mailto(char * /url*/)
        {
        }

        void
        BLWMLSession::addToHistory(BLHistoryItem * item)
        {
            mPageHistory.add(item);
            mCurrPage++;
            if((mPageHistory.getNumElements() > kHistorySize) && (mCurrPage != 0))

```

```

        BLHistoryItem * removedItem = mPageHistory.remove(0);
        delete removedItem;
        mCurrPage--;
        mLastPage--;
    }

    mLastNavigationAction = kGoTo;

    void
    BLWMLSession::goBack(void)
    {
        if(mCurrPage <= 0)
            return;

        mLastNavigationAction = kGoBack;
        loadPage(mPageHistory.getElementAt(--mCurrPage));
    }

    void
    BLWMLSession::goForward(void)
    {
        if((mCurrPage + 1) >= mPageHistory.getNumElements())
            return;

        mLastNavigationAction = kGoForward;
        loadPage(mPageHistory.getElementAt(++mCurrPage));
    }

    void
    BLWMLSession::goHome(void)
    {
        // only add a new home page to vector if not at the home page
        if(mCurrPage >= 1)
            goto(kDefaultPage, false);
    }

    void
    BLWMLSession::refresh(bool onlyErase)
    {
        if(mView)
        {
            if(onlyErase)
                mView->eraseView();
            else
                mView->newPage();
            mView->draw();
        }
    }

    CharPtr
    BLWMLSession::skipHTTP(CharPtr url)
    {
        if(StrNCompare(url, "http://", StrLen("http://")) == 0)
            return url + StrLen("http://");
        else
            return url;
    }

    void
    BLWMLSession::loadPage(BLHistoryItem * item)
    {
        CharPtr url = item->getURL();
        CharPtr cardID = item->getCardID();
    }

```

```

CharPtr currPageURL = NULL;
if (mDestCardID != NULL)
{
    delete [] mDestCardID;
    mDestCardID = NULL;
}
if (mLastPage >= 0) && (mLastPage < mPageHistory.getNumElements())
{
    BLHistoryItem * curPage = mPageHistory.getElementAt(mLastPage);
    if (curPage != NULL)
        currPageURL = curPage->getURL();
}
if (mParserP && (mParserP->getRoot() != NULL) &&
    currPageURL &&
    (StrCompare(skipHTTP(currPageURL), skipHTTP(url)) == 0))
{
    CharPtr activeID = mParserP->getRoot()->getActiveCardID();
    if (activeID && cardID && (StrCompare(cardID, activeID) == 0))
    {
        mPageHistory.remove(mCurrPage);
        mCurrPage--;
        return;
    }
    else
    {
        if ((cardID == NULL) && (activeID == NULL))
        {
            mPageHistory.remove(mCurrPage);
            mCurrPage--;
        }
        else if (mParserP->getRoot()->goToCard(cardID))
        {
            mView->newPage();
            mView->draw();
        }
        else
        {
            if (!isDone)
            {
                mDestCardID = BLUtils::cloneString(cardID);
                readFailure();
            }
            else
            {
                mLastPage = mCurrPage;
                return;
            }
        }
        mDestCardID = BLUtils::cloneString(cardID);
        mOldParserP = mParserP;
        mDownloadingImages = false;
        initiateDownload(BLUtils::cloneString(url));
    }
}
/*
returns port number from URL, also removes :port_num from
URL string, if present.
*/
CharPtr
BLMWLSession::extractPortNumber(CharPtr url)
{
    UInt i = 0;
    UInt colonPos = 0;
    CharPtr portNumberString = new Char(kMaxPortStringSize);
    StrPrintf(portNumberString, "%i", kDefaultPort);
}
}

while (true)
{
    if (url[i] == '/')
        return portNumberString;
    else if (url[i] == ':')
    {
        colonPos = i;
        break;
    }
    else if (url[i] == '\0')
        return portNumberString;
    i++;
}
if (colonPos != 0)
{
    i = colonPos + 1;
    while (BLUtils::isDigit(url[i]))
        i++;
    if (i == colonPos)
        return portNumberString;
    else if ((i - colonPos) >= kMaxPortStringSize)
        i = colonPos + kMaxPortStringSize - 1;
    StrNCopy(portNumberString, url[colonPos + 1], i - colonPos - 1);
    MemMove(url + colonPos, url + i, StrLen(url) + 1);
}
return portNumberString;
}

CharPtr
BLMWLSession::extractCardName(CharPtr url)
{
    int pos = (int) StrLen(url);
    while (url[pos] != '/') && (pos >= 0)
    {
        if ((url[pos] == '#') && (url[pos - 1] == '/'))
        {
            if (url[pos + 1])
            {
                CharPtr cardName = BLUtils::cloneString(url[pos + 1]);
                url[pos] = '\0';
                return cardName;
            }
        }
        pos--;
    }
    return NULL;
}

bool
BLMWLSession::initiatedDownload(CharPtr url)
{
    if (!BLPrefs::sPrefs.showImages && mDownloadingImages)
        return true;
    bool inCache = false;
    CharPtr originalURL = url;
    mNetworkingP->registerReader(this);
    while (*url && BLUtils::isWhitespace(*url))
        url++;
}

```



```

    mDownloadingImages = true;
    mCurrImage = mParserP->getImages().remove(pos--);
    if(mCurrImage != NULL)
    {
        mOldParserP = NULL;
        mContentLength = 0;
        mBytesUsed = 0;
        mChunkSize = 0;
        initiateDownload(url);
        downloadChosen = true;
    }
    else if(bytesUsed > 0)
    {
        image = mParserP->getImages().remove(pos--);
        imageFound = true;
        int urlLength = (int) StrLen(entry) + 1;
        urlLength += (int) StrLen(entry + urlLength) + 1;
        // word align:
        if(((ULong)entry + (ULong)urlLength) % 2) != 0)
            urlLength++;
        image->setImageData(entry + urlLength);
    }
}

if(imageFound && mView)
{
    mView->erase();
    mView->draw();
}

if(mParserP->getRoot())
    mParserP->getRoot()->onEnterEvent(mLastNavigationAction);
}

inCache = true;
else
{
    mDestURL = originalURL;
    mPendingDownload = true;
    CharPtr networkErr;

    if((!networkErr = mNetworkingP->openURL(fullURL, mDownloadingImages)))
    {
        FrmCustomAlert(CustomAlert, networkErr, "", "");
        mPendingDownload = false;
        if(mView)
            mView->draw();
    }
    if(!mDownloadingImages)
        mParserP = NULL;
    inCache = false;
}

delete [] fullURL;
return inCache;
}

```

```

void BLWMLSession::initializeSession()
{
    mCacheEntry = NULL;
    mBytesUsed = 0;
    mContentLength = 0;
    mChunkSize = 0;
    mDownloadingImages = false;
    if(mView)
        mView->setRoot(NULL);
    mOldParserP = NULL;
}

int BLWMLSession::readChunkSize()
{
    int skipBytes = 0;
    Char sizeBuffer(kChunkLengthBytes);
    if(mChunkSize == 0)
    {
        int bytesRead = mNetworkingP->peek(sizeBuffer, kChunkLengthBytes);
        if(bytesRead > 0)
        {
            int i;
            for(i=0; BLWUtils::isWhiteSpace(sizeBuffer[i]) || !sizeBuffer[i]; i++)
                break;
            if(i >= bytesRead)
            {
                mNetworkingP->readData(sizeBuffer, i);
                return i;
            }
            skipBytes += i;
            mChunkSize = BLWUtils::hexToInt(sizeBuffer(skipBytes), skipBytes,
            bytesRead-skipBytes);
            if(mChunkSize == -1)
            {
                mChunkSize = 0;
                return i;
            }
            mContentLength += mChunkSize;
            skipBytes += 2; // for the '\r\n'
        }
        if(skipBytes > 0)
            mNetworkingP->readData(sizeBuffer, skipBytes);
        return 0;
    }
    ULONG
    BLWMLSession::readData(CharPtr buffer, ULONG length)
    {
        BLNetworking::ContentType contentType = mNetworkingP->getContentType();
        if(mNetworkingP->getNetLibType() == BLNetworking::kNetLib_44 (length == 0))
            return 0;
        if(mOldParserP)
        {
            if(mParserP == mOldParserP)
                mParserP = NULL;
        }
    }
}

```

```

delete mOldParserP;
BIOFlushAll();
mCache.closeRecords();
mNetworking->clearTimer();
initializeSession();
}

if((contentType == BLNetworking::kWMMLC) && mDownloadingImages)
    return 0;

if((contentType == BLNetworking::kWMMLC) && (mParserP == NULL))
{
    mParserP = new WBXMLParser();
    if(mView)
        mView->newPage();
}

if((mPendingDownload != (mCacheEntry == NULL)) && mDestURL)
{
    mCacheEntry = mCache.newRecord(mPageHistory, mCurrPage);
    // no more cache room.
    if(mCacheEntry == NULL)
        return 0;

    ErrNonFatalDisplayIf(mDestURL == NULL, "No Dest URL defined");

    mURLLength = strlen(mDestURL) + 1;
    DmWrite(mCacheEntry, 0, mDestURL, mURLLength);

    CharPtr string;
    UInt length;

    CharPtr contentLoc = mNetworking->getContentLocation();

    if(contentLoc != NULL)
    {
        string = contentLoc;
        length = strlen(string) + 1;
        contentLoc = mDestURL;
    }
    else
    {
        string = "";
        length = 1;
        contentLoc = mDestURL;
    }

    if((!mDownloadingImages))
    {
        delete [] mContentLocation;
        mContentLocation = BUUtils::cloneString(contentLoc);

        DmWrite(mCacheEntry, mURLLength, string, length);

        mURLLength += length;

        // word align:
        if(((ULONG)mCacheEntry + (ULONG)mURLLength) % 2) != 0)
            mURLLength++;

        mPendingDownload = false;
    }
    else if(mDestURL == NULL)
        return 0;

    if(mCacheEntry == NULL)
        return 0;

    bool done = false;
    Sword bytesReceived;

    if(mNetworkingP->getNetLibType() == BLNetworking::kNetLib)
    {
        bytesReceived = (Sword) length;
        if(!mCache.growRecord((UShort)(mBytesUsed + mURLLength + length)))
            return 0;

        if(length > 0)
            DmWrite(mCacheEntry, mBytesUsed + mURLLength, buffer, length);
        else
            return 0;
    }
    else if(mNetworkingP->getNetLibType() == BLNetworking::kNetLib)
    {
        // headers are already parsed, we just need to read data directly
        // into a cache record:

        Word rcvLen = mCache.bytesLeft() - mURLLength;
        Err err = 0;

        if(rcvLen <= 0)
            return 0;
        if(mNetworkingP->getEncodingType() == HTTPRequest::kChunked)
        {
            if(readChunkSize() != 0)
                return 1;

            rcvLen = min(rcvLen, mChunkSize);
        }
        else if(rcvLen > 0)
        {
            bytesReceived =
                NetLibDmReceive(AppNetRefnum,
                               mNetworkingP->getSock(),
                               mCacheEntry,
                               mBytesUsed + mURLLength,
                               rcvLen, // flags
                               0, NULL, // sender info
                               -1,
                               &err);

            mChunkSize -= bytesReceived;
            if(!mCache.growRecord((UShort)(mBytesUsed + mURLLength + bytesReceived)))
                return 0;
            if(err != 0)
                return 0;
        }
        else
            return 0;

        if(bytesReceived > 0)
        {
            mBytesUsed += bytesReceived;

            if(contentType == BLNetworking::kWMMLC)
            {
                mCurrImage = NULL;
                mLastPage = mCurrPage;

                try
                {
                    if(mParserP->readData(mCacheEntry + mURLLength, mBytesUsed) == 0)
                    {
                        done = true;
                        if(mNetworkingP->getEncodingType() == HTTPRequest::kChunked)
                        {
                            Char buffer[5];
                            // read final 0 length chunk header:

```

```

        readChunkSize();
        mNetworkingP->readData(buffer, 5);
    }
    mView->setRoot(mParserP->getRoot());
    if(mDestCardID != NULL)
    {
        if(mParserP->getRoot()->goToCard(mDestCardID))
        {
            mView->draw();
            delete() mDestCardID;
            mDestCardID = NULL;
        }
        else if(done)
        {
            FrmAlert(DownloadErrorAlert);
            mView->draw();
        }
        else
            mView->draw();
    }
    catch(OutOfMemoryException e)
    {
        return 0;
    }
}
else if(contentType == BLNetworking::kPalBitmap)
{
    // !!! doesn't handle chunked
    if( (mBytesUsed >= mNetworkingP->getContentLength()) &&
        (mCurrImage != NULL) )
    {
        done = true;
        mCacheEntry = mCache.checkInCurrent(true);
        mCurrImage->setImageData(mCacheEntry + mURLLength);
        if(mView)
        {
            mView->eraseView();
            mView->draw();
        }
    }
}
else
{
    done = true;
}
if(done)
{
    mIsDone = true;
    bytesReceived = 0;
    if(!mParserP)
        return 0;
    if(contentType == BLNetworking::kMMLC)
    {
        if(!mDownloadingImages)
        {
            mCache.checkInCurrent(false);
        }
        else
            mCache.removeCurrent();
        bool gotImage = false;
        while(mParserP->getImages().getNumElements() > 0)
    {

```

```

        mDownloadingImages = true;
        mCurrImage = mParserP->getImages().remove(0);
        if(mCurrImage != NULL)
        {
            mOldParserP = NULL;
            CharPtr imageSource = mCurrImage->getSrc();
            if(mContentLocation && imageSource && mContentLocation[0] && imageSource[0])
            {
                CharPtr url = convertRelativeURL(mContentLocation, imageSource);
                mContentLength = 0;
                mBytesUsed = 0;
                mChunkSize = 0;
                if(!initiateDownload(url))
                {
                    if(gotImage)
                    {
                        if(mView)
                        {
                            mView->eraseView();
                            mView->draw();
                        }
                    }
                    return 1;
                }
                else
                {
                    gotImage = true;
                }
            }
        }
        if(gotImage && mView)
        {
            mView->eraseView();
            mView->draw();
        }
        mDownloadingImages = false;
        mOldParserP = mParserP;
        if(!mParserP->getRoot())
            mParserP->getRoot()->onEnterEvent(mLastNavigationAction);
    }
    return bytesReceived;
}

void
BLMMLSession::setTimer(ULong value, Timer * timerP)
{
    mNetworkingP->setTimer(value, timerP);
}

void
BLMMLSession::readFailure()
{
    if(!mDownloadingImages)
    {
        switch(mLastNavigationAction)
        {
            case kGoTo:
                delete mPageHistory.removeLast();
                mCurrPage--;
                break;
            case kGoBack:

```

```

mCurrPage++;
break;

case kGoForward:
mCurrPage--;
break;
)

FrmAlert(DownloadErrorAlert);

mView->draw();

mParserP = mOldParserP;
mOldParserP = NULL;

mLastNavigationAction = kNone;

)
// BLWMLSession.h
#pragma once
#include "BLURLHandler.h"
#include "BLTypes.h"
#include "BLWMLView.h"
#include "BLVector.h"
#include "BLNetworking.h"
#include "BLCacheDB.h"
#include "WMLVariables.h"

class BLWMLSession : public BLURLHandler, public DataReader
{
public:
    BLWMLSession(BLNetworking * networkingP);
    ~BLWMLSession();

    virtual void setView(BLWMLView * view) { mView = view; }
    virtual void getView() { return mView; }

    virtual void goto(CharPtr url, bool isRelativeURL);
    virtual void mailto(char * url);
    virtual void goBack();
    virtual void goForward();
    virtual void goHome();
    virtual void refresh(bool onlyFrame = false);
    virtual void initiateDownload(CharPtr url);
    virtual void readData(CharPtr buffer, ULONG length);
    virtual void readFailure();

    XMLParser *
        getParser() { return mParserP; }

    void
        addToHistory(BLHistoryItem * item);
    void
        loadPage(BLHistoryItem * item);

    void
        setTimer(ULONG value, Timer * timerP);

    void
        initializeSession();

    WMLVariables *
        getVars() { return mVars; }

protected:
    CharPtr
        skipHTTP(CharPtr url);

    CharPtr
        convertRelativeURL(CharPtr baseUrl, CharPtr relativeURL);

    void
        verifyHTTPLash(CharPtr url);

    CharPtr
        extractPortNumber(CharPtr url);

    CharPtr
        extractCardName(CharPtr url);

    int
        readChunkSize();

    BLWMLView * mView;
    int
        mCurrPage;
    int
        mLastPage;

    BLNetworking * mNetworkingP;
    WBXMLParser * mParserP;
    WBXMLParser * mOldParserP;

    WMLImage * mCurrImage;

    WMLVariables mVars;

    BLCacheDB mCache;
    CharPtr mContentLocation;
    CharPtr mCacheEntry;
    CharPtr mDestURL;
    CharPtr mDestCardID;
    ULONG mBytesUsed;
    bool mIsDone;
    mURLLength;
    int mContentLength;
    int mChunkSize;

    bool mDownloadingImages;
    bool mPendingDownload;

    NavigationAction mLastNavigationAction;
};
// BLWMLTable.cpp
#include "BLWMLTable.h"

const int kMaxLogicalWidth = 32000;
const int kMaxLogicalHeight = 32000;
const int kMaxColumns = 30;
const int kMinColSpace = 5;

WMLTable::WMLTable()
{
    mNumColumns = 0;
    mAlignArray = NULL;
    mTitle = NULL;

    mAlignArrayLength = 0;
}

WMLTable::WMLTable(BLGlyph * parent, const WMLTable &original)
: WMLTableElement(parent)
{
    mNumColumns = original.mNumColumns;
    mAlignArray = original.mAlignArray;
    mTitle = original.mTitle;

    if(mAlignArray)
        mAlignArrayLength = StrLen(mAlignArray);
    else
        mAlignArrayLength = 0;

    if((mNumColumns > 0) && (mNumColumns < kMaxColumns))
        mColWidths = (ColumnMetrics *) sMemManager.newChunk(
            (sizeof(ColumnMetrics) * mNumColumns));
    else
        mColWidths = NULL;
}

bool
WMLTable::flow(BLParentGlyph * parent, const Point &areaExtent)
{

```



10428800

```

    if(!computeColWidths(areaExtent.x))
        return false;

    if(mTitle)
    {
        TextState plainState;
        WMLLine * line = new WMLLine(parent, kCenter);
        WMLTextRun * run = new WMLTextRun( line,
            mTitle,
            (int) StrLen(mTitle),
            plainState);

        int tableWidth = 0;
        for(int i=0; i<mNumColumns; i++)
        {
            tableWidth += mColWidths[i].allocated;
        }
        run->computeMetrics(tableWidth);
        line->computeMetrics(tableWidth);
    }

    BLIterator<BLGlyph*, kChildrenInc> iter(*children());
    while(iter.hasNext())
    {
        // iterator over table rows:
        WMLTableRow * row = dynamic_cast<WMLTableRow*>(iter.getNext());
        parent->children()->add(row);
        BLIterator<BLGlyph*, kChildrenInc> rowIter(*row->children());
        while(rowIter.hasNext())
        {
            int columnNum = 0;
            while(rowIter.hasNext())
            {
                // iterator over cells in row:
                WMLTableCell * cell = dynamic_cast<WMLTableCell*>(rowIter.getNext());
                if(columnNum < mAlignArrayLength)
                {
                    switch(mAlignArray[columnNum])
                    {
                        case 'L':
                        case 'D':
                            cell->setAlignment(kLeft);
                            break;
                        case 'C':
                            cell->setAlignment(kCenter);
                            break;
                        case 'R':
                            cell->setAlignment(kRight);
                            break;
                    }
                }
                Point cellPoint(min(areaExtent.x, mColWidths[columnNum].allocated),
                    kMaxLogicalHeight);
                if(!cell->flow(cellPoint))
                    return false;
                cell->computeMetrics(mColWidths[columnNum].allocated);
                columnNum++;
                if(columnNum == mNumColumns)
                {
                    columnNum--;
                    row->computeMetrics(areaExtent.x);
                }
            }
        }
    }

    bool
    WMLTable::computeColWidths(int availableWidth)
    {
        int columnNum;
        if(mColWidths == NULL)
            return false;
        for(int i=0; i<mNumColumns; i++)
        {
            mColWidths[i].min = 0;
            mColWidths[i].req = 0;
            mColWidths[i].allocated = 0;
        }
        BLIterator<BLGlyph*, kChildrenInc> iter(*children());
        while(iter.hasNext())
        {
            // iterator over table rows:
            BLGlyph * row = iter.getNext();
            if(dynamic_cast<WMLTableRow*>(row) == NULL)
            {
                // badly formed table...
                return false;
            }
            BLIterator<BLGlyph*, kChildrenInc> rowIter(*row->children());
            columnNum = 0;
            while(rowIter.hasNext())
            {
                // iterator over cells in row:
                WMLTableCell * cell = dynamic_cast<WMLTableCell*>(rowIter.getNext());
                if(cell == NULL)
                {
                    // badly formed table...
                    return false;
                }
                cell->computeWidth();
                mColWidths[columnNum].setMin(cell->mMinWidth);
                mColWidths[columnNum].setReq(cell->mReqWidth);
                columnNum++;
                if(columnNum == mNumColumns)
                    columnNum--;
            }
            int leftOver = availableWidth;
            int totalGap = 0;
            for(int i=0; i<mNumColumns; i++)
            {
                mColWidths[i].allocated = mColWidths[i].min;
                leftOver -= mColWidths[i].allocated;
                totalGap += mColWidths[i].req - mColWidths[i].min;
            }
            if((leftOver > 0) && (totalGap > 0))
            {
                for(int i=0; i<mNumColumns; i++)
                {
                    mColWidths[i].allocated += (leftOver*(100*mColWidths[i].req - mColWidths[i].min)
                        / totalGap) / 100;
                }
            }
        }
    }

```

```

// set widths for table cells
while (iter2.hasNext())
{
    // iterator over table rows:
    BLGlyph * row = iter2.getNext();
    BLIterator<BLGlyph*, kChildrenInc> rowIter(*row->children());
    columnNum = 0;
    while (rowIter.hasNext())
    {
        // iterator over cells in row:
        WMLTableCell * cell = dynamic_cast<WMLTableCell*> (rowIter.getNext());
        LineMetrics metrics = cell->getMetrics();
        metrics.width = mColWidths[columnNum].allocated;
        cell->setMetrics(metrics);
        columnNum++;
        if (columnNum == mNumColumns)
            columnNum--;
    }
    return true;
}

void WMLTable::draw(const RectAngleType & /*displayArea*/, const Point & /*offset*/)
{
}

void WMLTable::computeMetrics(int /*lineWidth*/)
{
}

WMLTableRow::WMLTableRow(BLGlyph * parent)
: WMLLine(parent, kLeft)
{
}

WMLTableCell::WMLTableCell(BLGlyph * parent)
: WMLLineElement(), WMLParagraph(parent, WMLParagraph())
{
    mMinWidth = 0;
    mReqWidth = 0;
}

void WMLTableCell::computeMetrics(int /*lineWidth*/)
{
    mMetrics.baselineOffset = 0;
    mMetrics.overhang = getExtent().y;
}

void WMLTableCell::computeWidth()
{
    BLIteratorNode<BLGlyph*, kChildrenInc> * curIter;
    BLIteratorNode<BLGlyph*, kChildrenInc> * iter;
}

```

```

        iter = new BLIteratorNode<BLGlyph*, kChildrenInc>(mChildren);
        BLStack iterStack;
        iterStack.push(iter);
        while (iterStack.top() != NULL)
        {
            curIter = (BLIteratorNode<BLGlyph*, kChildrenInc> *) iterStack.pop();
            while (curIter->hasNext())
            {
                BLGlyph * elem = curIter->getNext();
                if (elem->children())
                {
                    iter = new BLIteratorNode<BLGlyph*, kChildrenInc> (*elem->children());
                    if (curIter->hasNext())
                    {
                        iterStack.push(curIter);
                        curIter = NULL; // make it NULL so it won't be deleted
                    }
                    iterStack.push(iter);
                    break;
                }
                else
                {
                    if (dynamic_cast<WMLTextRun*> (elem) != NULL)
                    {
                        WMLTextRun * textRun = dynamic_cast<WMLTextRun*> (elem);
                        TextState state = textRun->getState();
                        state.enable();
                        textRun->computeMetrics(kMaxLogicalWidth);
                        setReqWidth(textRun->getMetrics().width);
                        if (state.bits.WordWrap)
                            setMinWidth(textRun->findLongestWord());
                        else
                            setMinWidth(textRun->getMetrics().width);
                    }
                    else if (dynamic_cast<WMLImage*> (elem) != NULL)
                    {
                        WMLImage * image = dynamic_cast<WMLImage*> (elem);
                        Point extent = image->getExtent();
                        mReqWidth += extent.x;
                        setMinWidth(extent.x);
                    }
                }
            }
            delete curIter;
        }
    }
}

// BLMMLTable.h
#include "BLGlyph.h"
struct ColumnMetrics
{
    int min;
    int req;
    int allocated;
    void setMin(int newMin) { min = max(newMin, min); }
    void setReq(int newReq) { req = max(newReq, req); }
};

class WMLTable : public WMLLineElement

```

```

    public:
        WMLTable();
        WMLTable(BLGlyph * parent, const WMLTable &original);

        virtual bool flow(BLParentGlyph * parent, const Point &areaExtent);
        virtual void draw(const RectangleType & displayArea, const Point &offset);
        virtual void computeMetrics(int lineWidth);
        virtual Point getExtent() { return Point::zeroPoint(); }
        virtual Point getExtent(const Point &areaExtent) { return Point::zeroPoint(); }

        virtual BLVector<BLGlyph*, kChildrenInc> * children()
        { return &children; }

        virtual LineMetrics getMetrics() { return mMetrics; }
        virtual void setMetrics(const LineMetrics &metrics) { mMetrics = metrics; }

        friend class WBXMLParser;

    protected:
        BLVector<BLGlyph*, kChildrenInc> mChildren;
        bool computeColWidths(int availableWidth);

        int mNumColumns;
        ColumnMetrics * mColWidths;
        int mTotalWidth;

        CharPtr mTitle;
        CharPtr mAlignArray;
        UInt mAlignArrayLength;

        LineMetrics mMetrics;

};

class WMLTableRow : public WMLLine
{
    public:
        WMLTableRow(BLGlyph * parent);

        virtual void draw(const RectangleType & displayArea, const Point &offset)
        { WMLLine::draw(displayArea, offset); }

};

// ...! you really want a virtual base class, right?
class WMLTableCell : public WMLParagraph, virtual public WMLLineElement
{
    public:
        WMLTableCell(BLGlyph * parent);

        void computeWidth();

        virtual void computeMetrics(int lineWidth);

        virtual Point getExtent() { return WMLParagraph::getExtent(); }
        virtual Point getExtent(const Point &areaExtent)
        { return WMLParagraph::getExtent(areaExtent); }

        virtual void draw(const RectangleType & displayArea, const Point &offset)
        { return WMLParagraph::draw(displayArea, offset); }

        virtual bool click(const RectangleType & displayArea,
                           const Point &offset,
                           PointType p)
        { return WMLParagraph::click(displayArea, offset, p); }

        virtual void setAlignment(Alignment align) { mAlign = align; }
};

```

```

    virtual void setMetrics(const LineMetrics &metrics) { mMetrics = metrics; }

    int mMinWidth;
    int mReqWidth;

    protected:
        void setMinWidth(int newMin) { mMinWidth = max(mMinWidth, newMin); }
        void setReqWidth(int newMin) { mReqWidth = max(mReqWidth, newMin); }

        LineMetrics mMetrics;

};

// BLWMLView.cpp
#include "BLWMLView.h"
#include "BLUtlis.h"
#include "BLWMLSession.h"
#include "BLWMLActions.h"
#include "BLWMLForm.h"

const int kLineSize = 11;
const int kColumnSize = 11;
const int kScrollbarHeight = 7;

BLWMLView::BLWMLView(
    formP,
    BLWMLSession * urlHandler,
    RectangleType rect,
    vScrollbarID,
    hScrollbarID,
    UInt
    UInt
) : mRootExtent(0, 0)
{
    mFormP = formP;
    mParser = NULL;
    mURLHandler = urlHandler;
    mRect = rect;
    mVScrollbarID = vScrollbarID;
    mHScrollbarID = hScrollbarID;
    mVScrollbarP = (ScrollbarPtr) BLUtlis::getObjectPtr(mVScrollbarID);
    mHScrollbarP = (ScrollbarPtr) BLUtlis::getObjectPtr(mHScrollbarID);
    mIsMoving = false;
    mIsMouseInView = false;
    initPage();

    mRoot = NULL;

}

BLWMLView::~BLWMLView()
{
    BLGlyph::flushAll();
    delete mParser;

}

void
BLWMLView::open()
{
    mURLHandler->goTo(kDefaultPage, NULL);
}

void
BLWMLView::scrollBarEvent(void * pScrollbar)
{
    if(pScrollbar == mVScrollbarP)

```

```

Short min, max, pageSize;
Short vValue, hValue;

```

```

    if (mVScrollBar->attr.activeRegion == sclDownArrow)
        scrollVert(kLineStyle - 1);
    else if (mVScrollBar->attr.activeRegion == sclUpArrow)
        scrollVert(-(kLineStyle - 1));
    else
        draw();
    else if (pScrollBar == mHScrollBar)
    {
        if (mHScrollBar->attr.activeRegion == sclDownArrow)
            scrollHoriz(kColumnSize - 1);
        else if (mHScrollBar->attr.activeRegion == sclUpArrow)
            scrollHoriz(-(kColumnSize - 1));
        else
            draw();
    }

void
BLMWMLView::scrollHoriz(int scrollPixels)
{
    Short hValue, newHValue;
    Short min, max, pageSize;
    SclGetScrollBar(mHScrollBar, &hValue, &min, &max, &pageSize);
    newHValue = hValue + scrollPixels;
    if (newHValue < min)
        newHValue = min;
    if (newHValue > max)
        newHValue = max;
    if (newHValue != hValue)
    {
        SclSetScrollBar(mHScrollBar, newHValue, min, max, pageSize);
        draw();
    }
}

void
BLMWMLView::scrollVert(int scrollPixels)
{
    Short vValue, newVValue;
    Short min, max, pageSize;
    SclGetScrollBar(mVScrollBar, &vValue, &min, &max, &pageSize);
    newVValue = vValue + scrollPixels;
    if (newVValue < min)
        newVValue = min;
    if (newVValue > max)
        newVValue = max;
    if (newVValue != vValue)
    {
        SclSetScrollBar(mVScrollBar, newVValue, min, max, pageSize);
        draw();
    }
}

Boolean
BLMWMLView::handleEvent(EventPtr eventP)
{
    Boolean handled = false;

```

```

Short min, max, pageSize;
Short vValue, hValue;
Point mousePoint;
Point offset;
switch(eventP->eType)
{
    case sclRepeatEvent:
        scrollBarEvent(eventP->data.sclRepeat.pScrollBar);
        break;
    case menuEvent:
        break;
    case penMoveEvent:
        mousePoint = Point(eventP->screenX, eventP->screenY);
        if (BLGlyph::sClickedAction != NULL)
        {
            if (!BLUtils::isInsideRectVector(mousePoint, BLGlyph::sActiveRectVector))
            {
                // (un)highlight link
                BLGlyph::sHighlightAction = BLGlyph::kUnhighlight;
                SclGetScrollBar(mHScrollBar, &hValue, &min, &max, &pageSize);
                SclGetScrollBar(mVScrollBar, &vValue, &min, &max, &pageSize);
                offset = Point((SWord) hValue,
                               (SWord) vValue);
                WinSetClip(&offset);
                mRoot->click(&offset, mousePoint);
                WinResetClip();
                BLGlyph::sClickedAction = NULL;
                BLGlyph::sActiveRectVector.removeAll();
                mPenDownPoint.x = mousePoint.x;
                mPenDownPoint.y = mousePoint.y;
            }
        }
        if (BLGlyph::sClickedAction == NULL) && mIsMouseInView)
        {
            bool redraw = false;
            Point scrollPixels(-(mousePoint.x - mPenDownPoint.x),
                               -(mousePoint.y - mPenDownPoint.y));
            if (mIsHorizShowing)
            {
                // Horizontal:
                if (!mIsMoving)
                {
                    mPixelsMoved.x += scrollPixels.x;
                    if (BLUtils::abs(mPixelsMoved.x) > kColumnSize/2)
                    {
                        scrollPixels.x = mPixelsMoved.x;
                        mIsMoving = true;
                    }
                    else
                        scrollPixels.x = 0;
                }
            }
            if ((scrollPixels.x != 0) &&
                (BLUtils::abs(3*scrollPixels.x) > BLUtils::abs(scrollPixels.y)))
            {
                scrollHoriz(scrollPixels.x);
            }
            if (mIsVertShowing)
            {
                // Vertical:
                if (!mIsMoving)
                {

```

```

File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM
    mPixelMoved.y += scrollPixels.y;
    if (BLUtils::abs(mPixelMoved.y) > kLineStyle/2)
    {
        scrollPixels.y = mPixelMoved.y;
        mIsMoving = true;
    }
    else
        scrollPixels.y = 0;
}
if ((scrollPixels.y != 0) &&
    (BLUtils::abs(3*scrollPixels.y) > BLUtils::abs(scrollPixels.x)))
{
    scrollVert(scrollPixels.y);
}

mPenDownPoint = mousePoint;
handled = true;
break;

case penDownEvent:
    mousePoint = Point(eventP->screenX, eventP->screenY);
    if (mRoot == NULL)
        break;
    if (mousePoint.isInside(mRect))
    {
        mIsMouseInView = true;
        mPenDownPoint = Point(eventP->screenX, eventP->screenY);
        ScrollGetScrollBar(mHScrollBarP, hValue, fmin, fmax, fpageSize);
        ScrollGetScrollBar(mVScrollBarP, vValue, fmin, fmax, fpageSize);
        offset = Point((SWord) hValue,
                       (SWord) vValue);
        WinSetClip(fmRect);
        // first pass: find clicked link
        BLGlyph::sHiliteAction = BLGlyph::kFindSelection;
        mRoot->click(mRect, offset, mousePoint);
        // hilite link
        BLGlyph::sHiliteAction = BLGlyph::kHilite;
        mRoot->click(mRect, offset, mousePoint);
        WinResetClip();
        handled = true;
        break;
    }
case penUpEvent:
    mousePoint = Point(eventP->screenX, eventP->screenY);
    mIsMouseInView = false;
    if (BLGlyph::sClickedAction != NULL)
    {
        // (un)hilite link
        ScrollGetScrollBar(mHScrollBarP, hValue, fmin, fmax, fpageSize);
        ScrollGetScrollBar(mVScrollBarP, vValue, fmin, fmax, fpageSize);
        offset = Point((SWord) hValue,
                       (SWord) vValue);
        WinSetClip(fmRect);
        BLGlyph::sHiliteAction = BLGlyph::kUnhilite;
    }
}

mRoot->click(mRect, offset, mousePoint);
BLGlyph::sClickedAction->activate();
BLGlyph::sClickedAction = NULL;
BLGlyph::sActiveRectVector.removeAll();
WinResetClip();
handled = true;
}
else
{
    mIsMoving = false;
    mPixelMoved = Point(0, 0);
    break;
}

case keyDownEvent:
    if (eventP->data.keyDown.chr == pageDownChr)
    {
        ScrollGetScrollBar(mVScrollBarP, vValue, fmin, fmax, fpageSize);
        if (vValue < max)
        {
            vValue += pageSize - kLineStyle;
            if (vValue > max)
                vValue = max;
            ScrollGetScrollBar(mVScrollBarP, vValue, min, max, pageSize);
            draw();
        }
        handled = true;
    }
    else if (eventP->data.keyDown.chr == pageUpChr)
    {
        ScrollGetScrollBar(mVScrollBarP, vValue, fmin, fmax, fpageSize);
        if (vValue > min)
        {
            vValue -= pageSize - kLineStyle;
            if (vValue < min)
                vValue = min;
            ScrollGetScrollBar(mVScrollBarP, vValue, min, max, pageSize);
            draw();
        }
        handled = true;
        break;
    }
    return handled;
}

RectangleType
BLMWLVView::getBounds()
{
    // !!! oops...won't be correct if horiz scrollbar present...
    return mRect;
}

void
BLMWLVView::draw()
{
    if (mRoot == NULL)
        return;
    Short vValue, hValue, unused;
    FrmPtr activeFormP = FrmGetActiveForm();

```

```

    }
    WinSetRectAngle(&displayRect, 0);
}

Point extent;
try
{
    WinSetClip(&mRect);
    mRoot->draw(displayRect, offset);
    WinResetClip();
    extent = mRoot->getExtent(mRect.extent);
}
catch (OutOfMemoryException e)
{
    WinResetClip();
    goto DRAW_END;
}

if (mRootExtent != extent)
{
    mRootExtent = extent;
    if (extent.x > displayRect.extent.x)
    {
        if (!mIsHorizShowing)
        {
            mRect.extent.y -= kScrollbarHeight;
            mIsHorizShowing = true;
            RectangleType bounds;
            FrmGetObjectBounds(activeFormP, FrmGetObjectIndex(activeFormP,
mHScrollbarBarID), &bounds);
            WinEraseRectangle(&bounds, 0);
            FrmShowObject(activeFormP, FrmGetObjectIndex(activeFormP, mHScrollbarBarID));
        }
        int pageMax = extent.x - mRect.extent.x;
        if (pageMax < 0)
            pageMax = 0;
        ScrollScrollbar(mHScrollbarBarP, hValue, 0,
            pageMax,
            mRect.extent.x);
    }
    if (extent.y > displayRect.extent.y)
    {
        if (!mIsVertShowing)
        {
            FrmShowObject(activeFormP, FrmGetObjectIndex(activeFormP, mVScrollbarBarID));
            mIsVertShowing = true;
        }
        int pageMax = extent.y - mRect.extent.y;
        if (pageMax < 0)
            pageMax = 0;
        ScrollScrollbar(mVScrollbarBarP, vValue, 0,
            pageMax,
            mRect.extent.y);
    }
}

DRAW_END:

```

```

File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM
TextState::enableBasicState();
mLastVScrollValue = vValue;
mLastHScrollValue = hValue;

void
BLMWMLView::newPage()
{
    FormPtr activeFormP = FrmGetActiveForm();
    WinResetClip();
    if(mIsHorizShowing)
    {
        mRect.extent.y += kScrollbarHeight;
        FrmHideObject(activeFormP, FrmGetObjectIndex(activeFormP, mHScrollbarID));
    }
    if(mIsVertShowing)
    {
        FrmHideObject(activeFormP, FrmGetObjectIndex(activeFormP, mVScrollbarID));
    }
    initPage();
    mRootExtent = Point::sZeroPoint();
    eraseView();
}

void
BLMWMLView::eraseView()
{
    WinEraseRectangle(mRect, 0);
}

void
BLMWMLView::initPage()
{
    ScrollBarPtr (ScrollBarPtr) BLUtils::getObjectPtr(mVScrollbarID, 0, 0, 0, 0);
    ScrollBarPtr (ScrollBarPtr) BLUtils::getObjectPtr(mHScrollbarID, 0, 0, 0, 0);
    mLastHScrollValue = 0;
    mLastVScrollValue = 0;
    mIsHorizShowing = false;
    mIsVertShowing = false;
    BLGlyph::isClickedAction = NULL;
    BLGlyph::isActiveRectVector.removeAll();
}

void
BLMWMLView::refreshFormPointer()
{
    mFormP->refreshFormPointer();
    mVScrollbarP = (ScrollBarPtr) BLUtils::getObjectPtr(mVScrollbarID);
    mHScrollbarP = (ScrollBarPtr) BLUtils::getObjectPtr(mHScrollbarID);
}

// BLMWMLView.h
#pragma once
#include "BLView.h"
#include "WBXMLParser.h"
#include "BLURLHandler.h"
#include "BLGlyph.h"

```

```

#include "BLTypes.h"
#include "BLPageDP.h"
class BLWMLSession;
class BLWMLForm;
class BLMWMLView : public BLView
{
public:
    BLMWMLView( BLWMLForm * formP,
                BLWMLSession * urlHandler,
                RectangleType rect,
                UInt vScrollbarID,
                UInt hScrollbarID);
    ~BLMWMLView();
    virtual void open();
    virtual void handleEvent(EventPtr eventP);
    virtual Boolean getBounds();
    virtual void draw();
    virtual void newPage();
    virtual void initPage();
    virtual void setRoot(WMLRoot * root) { mRoot = root; }
    virtual void eraseView();
    virtual void refreshFormPointer();

protected:
    void scrollBarEvent(void * pScrollbar);
    void scrollVert(int scrollPixels);
    void scrollHoriz(int scrollPixels);
    Short mLastHScrollValue;
    Short mLastVScrollValue;
    BLWMLForm * mFormP;
    WBXMLParser * mParser;
    WMLRoot * mRoot;
    BLWMLSession * mURLHandler;
    RectangleType mRect;
    UInt mVScrollbarID;
    UInt mHScrollbarID;
    ScrollBarPtr mVScrollbarP;
    ScrollBarPtr mHScrollbarP;
    Point mRootExtent;
    bool mIsHorizShowing;
    bool mIsVertShowing;
    Point mExtraPixels;
    Point mPenDownPoint;
    Point mPixelsMoved;
    bool mIsMoving;
    bool mIsMouseDownView;
    // BLWMLForm.cpp
#include "BLWMLForm.h"
BLWMLForm:BLWMLForm( UInt supportedFormID,
                     UInt viewGadgetID,
                     UInt scrollbarID)

```





File: G:\AllBrowserSource.c 06/15/2000, 11:00:18PM

```

XML_SetUserData(mParser, this);
XML_SetElementHandler( mParser,
    sStartElementHandler,
    sEndElementHandler );

XML_SetCharacterDataHandler(mParser,
    sCharacterDataHandler);

}

BLXMLParser::~BLXMLParser()
{
    XML_ParsetFree(mParser);
}

XMLElement *
BLXMLParser::parse()
{
    BLVector<XMLElement> *,
    XMLElement::kChildrenInc>::setMemManager(&XMLElement::sMemManager);
    BLVector<XMLAttribute, XMLElement::kAttrInc>::setMemManager(&XMLElement::sMemManager);

    if((mFile == NULL) || (mFileLength == 0))
        return NULL;

    int error = 1;

    if(!CachedPage)
        error = XML_Parse(mParser, mFile, (int) mFileLength, true);

    if (error == 0)
    {
        ErrDisplay(reportError());
        return NULL;
    }
    else return mRoot;
}

void
BLXMLParser::read(char * buffer, ULONG length)
{
    int error = XML_Parse(mParser, buffer, (int) length, false);

    if (error == 0)
    {
        ErrDisplay(reportError());
    }
}

char *
BLXMLParser::reportError()
{
    XML_Error error = XML_GetErrorCode(mParser);
    char * message;

    switch (error) {
        case XML_ERROR_NONE:
            message = "None";
            break;
        case XML_ERROR_NO_MEMORY:
            message = "No Memory";
            break;
        case XML_ERROR_SYNTAX:
            message = "Syntax Error";
            break;
        case XML_ERROR_NO_ELEMENTS:
            message = "No Elements";
            break;
        case XML_ERROR_INVALID_TOKEN:
            message = "Invalid Token";
            break;
    }

    BLXMLParser * t = (BLXMLParser *) userData;
    XML_Char **attsPtr = (XML_Char**) atts;
    XML_Char *attName, *attValue;
    XMLElement * parent = (XMLElement *) t->mElements.top();
    XMLElement * newNode = new XMLElement(parent, name);

    if (parent)
        parent->mChildren.add(newNode);
    else
        t->mRoot = newNode;

    return message;
}

void
BLXMLParser::StartElementHandler( void *userData,
    const XML_Char *name,
    const XML_Char **atts)
{
    BLXMLParser * t = (BLXMLParser *) userData;
    XML_Char **attsPtr = (XML_Char**) atts;
    XML_Char *attName, *attValue;
    XMLElement * parent = (XMLElement *) t->mElements.top();
    XMLElement * newNode = new XMLElement(parent, name);

    if (parent)
        parent->mChildren.add(newNode);
    else
        t->mRoot = newNode;

    return message;
}

```

```

while(*attsPtr != 0)
{
    attName = *attsPtr++;
    attValue = *attsPtr;
    newNode->addAttribute(attName, attValue);
    attsPtr++;
}

t->mElements.push(newNode);

void
BLXMLParser::sEndElementHandler(    void *userData,
const XML_Char *s, int len)
{
    BLXMLParser * t = (BLXMLParser *) userData;

    t->mElements.pop();
    // At some point, incremental rendering could work by
    // seeing if a card element was just popped.
}

```

```

void
BLXMLParser::sCharacterDataHandler(    void *userData,
const XML_Char *s,
int len)
{
    BLXMLParser * t = (BLXMLParser *) userData;

    XML_Element * parent = (XML_Element *) t->mElements.top();
    XMLTextNode * textNode = new XMLTextNode(parent, s, len);
    parent->mChildren.add(textNode);
}

// BLXMLParser.h

#pragma once

#include "XMLParser.h"
#include "BLStack.h"
#include "BLVector.h"
#include "BLTypes.h"
#include "BLMemoryManager.h"

class XMLAttribute
{
public:
    XMLAttribute()
        : mName(NULL), mValue(NULL) {}

    XMLAttribute(    const XML_Char * name,
                    const XML_Char * value);

    XML_Char * mName;
    XML_Char * mValue;
}

```

```

class XML_Element : public BLNode
{
public:
    enum { kChildrenInc = 4, kAttrInc = 2 };

    XML_Element( XML_Element * parent,
const XML_Char * name);

    virtual void
addAttribute(    const XML_Char * name,
const XML_Char * value);
}

```

```

// BLXMLParser.cpp

void *
operator new(size_t), size_t);
void
operator delete(void *, size_t);

XML_Element *
BLVector<XML_Element, kChildrenInc>
mChildren;
BLVector<XML_Attribute, kAttrInc>
mAttributes;

XML_Char * mName;

static void
flushAll() { sMemManager.flushAll(); }

static BLMemoryManager sMemManager;

class XMLTextNode : public XML_Element
{
public:
    XMLTextNode(XML_Element * parent,
const XML_Char * data,
int len);

    XML_Char * mText;
    int mLength;

};

class BLXMLParser : public DataReader
{
public:
    BLXMLParser(RawPageData * rawPage);
    ~BLXMLParser();

    virtual
    XML_Element * parse();
    virtual char * reportError();
    virtual void read(char * buffer, ULONG length);

protected:
    char * mFile;
    UInt mFileLength;

    XML_Parser mParser;
    BLStack mElements;
    XML_Element * mRoot;
    bool mCachedPage;

    static void
sStartElementHandler(    void *userData, *name,
const XML_Char **atts);

    static void
sEndElementHandler(    void *userData,
const XML_Char *name);

    static void
sCharacterDataHandler(    void *userData,
const XML_Char *s,
int len);

}; // BLXMLView.cpp

#include "BLXMLView.h"

BLXMLView::BLXMLView(    BLURLHandler * urlHandler,
RectanglePtr rect,
xmlData,
CharPtr
int
dataLength,
ScrollBarPtr
scrollBar)
: mParser(xmlData, dataLength)

```

```

{
    mURLHandler = urlHandler;
    mRect = rect;
    mScrollBar = scrollBar;
    mRoot = NULL;
}

```

```

BLXMLView::~BLXMLView()
{
}

```

```

Boolean
BLXMLView::handleEvent(EventPtr eventP)
{
    Boolean handled = false;
    return handled;
}

```

```

void
BLXMLView::draw()
{
    if(mRoot == NULL)
        layout();
}

void
BLXMLView::layout()
{
    mRoot = mParser.parse();
}

```

```

// BLXMLView.h

```

```

#include "BLView.h"
#include "BLXMLParser.h"
#include "BLURLHandler.h"
#include "BLGlyph.h"

```

```

class BLXMLView : public BLView
{
public:

```

```

    BLXMLView( BLURLHandler * urlHandler,
               RectanglePtr rect,
               CharPtr xmlData,
               int dataLength,
               ScrollBarPtr scrollBar);

```

```

    virtual ~BLXMLView();
    virtual Boolean handleEvent(EventPtr eventP);
    virtual void draw();
    virtual void layout();

```

```

protected:

```

```

    BLXMLParser mParser;
    XMLElement * mRoot;

    BLURLHandler * mURLHandler;
    RectanglePtr mRect;
    ScrollBarPtr mScrollBar;
}; // BrowserApp.cpp

#include "BrowserApp.h"
#include "BLPrefs.h"

```

```

BrowserApp::BrowserApp()
{

```

```

    mMmlForm = NULL;
}

```

```

BrowserApp::~BrowserApp()
{
    delete mMmlForm;
}

```

```

DWORD
BrowserApp::launchNormal()
{
    /* Byte *LPICF = (Byte *) 0xFFFFFFFF20;
    Byte *LVPW = (Byte *) 0xFFFFFFFFA05;

    *LPICF |= 0x02;
    *LVPW <= 2; */

    initialize();
    if(!BLUtils::checkExpirationDate())
        return 0;
}

```

```

BLGlyph::setMemoryManagers();

```

```

BLPrefs::read();

```

```

mMmlForm = new BLMMLForm(amNetwork, MainForm, MainViewGadget, MainVertScrollBar,
MainHorizScrollBar);
registerHandler(mMmlForm);

```

```

startNetworking();

```

```

FrmGotoForm(MainForm);

```

```

DWORD returnVal = eventLoop();

```

```

stopNetworking();

```

```

BLPrefs::write();

```

```

// *LVPW >= 2;

```

```

// *LPICF &= 0xF0;

```

```

return returnVal;

```

```

// BrowserApp.h

```

```

#include "BLApplication.h"
#include "BLMMLForm.h"
#include "BLBookmark.h"

```

```

class BrowserApp : public BLApplication
{
public:

```

```

    enum { appFileCreator = 'biki', kPrefsVersion = 0x1 };

```

```

    BrowserApp();

```

```

    ~BrowserApp();

```

```

    virtual DWORD launchNormal();

```

```

protected:

```

```

    BLMMLForm * mMmlForm;

```

```

}; // Header generated by Constructor for PalmOS 1.2

```

```

// Generated at 9:43:55 PM on Tuesday, June 13, 2000

```

```

// Generated for file: G:\Code\blueark\app\browser\src\BrowserRsrc.rsrc

```

```

// THIS IS AN AUTOMATICALLY GENERATED HEADER FILE FROM CONSTRUCTOR FOR PALMOS;
// DO NOT EDIT - CHANGES MADE TO THIS FILE WILL BE LOST
// Palm App Name: "Blazer"
// Palm App Version: "1.0"

```

```

// Resource: tFRM 2537
#define BookmarksForm
Width = 156, Height = 156, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button ID = 0
#define BookmarksEditOKButton
Width = 25, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditCancelButton
Width = 36, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditDeleteButton
Width = 35, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditNewButton
Width = 35, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditList
Width = 140, Usable = 1, Font = Standard, Visible Items = 10
// Resource: tFRM 2517
#define BookmarksAddForm
Width = 156, Height = 118, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button ID = 0

```

```

// Resource: tFRM 2530
#define BookmarksAddForm
Width = 156, Height = 118, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button ID = 0

```

```

// Resource: tFRM 2527
#define BookmarksDetailsForm
Width = 156, Height = 118, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button ID = 0
#define BookmarksDetailsOKButton
Width = 25, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksDetailsCancelButton
Width = 36, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksDetailsURLField
Width = 119, Height = 37, Usable = 1, Single Line = 0, Dynamic Size = 0, Left Justified = 1, Max Characters = 120, Font = Standard, Auto Shift = 0, Has Scroll Bar = 0, Numeric = 0
#define BookmarksDetailsURLField
Width = 119, Height = 37, Usable = 1, Single Line = 0, Dynamic Size = 0, Left Justified = 1, Max Characters = 120, Font = Standard, Auto Shift = 0, Has Scroll Bar = 0, Numeric = 0
#define BookmarksDetailsURLField
Width = 119, Height = 37, Usable = 1, Single Line = 0, Dynamic Size = 0, Left Justified = 1, Max Characters = 120, Font = Standard, Auto Shift = 0, Has Scroll Bar = 0, Numeric = 0

```

```

// Resource: tFRM 2507
#define BookmarksEditForm
Width = 156, Height = 156, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button ID = 0
#define BookmarksEditOKButton
Width = 25, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditCancelButton
Width = 36, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditDeleteButton
Width = 35, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard
#define BookmarksEditNewButton
Width = 35, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Non-bold Frame = 1, Font = Standard

```

```

Standard)
#define MainRightArrowButton 1007//((Left Origin = 134, Top Origin = 0,
Width = 10, Height = 13, Usable = 1, Anchor Left = 1, Anchor Right = 0,
Standard)
#define MainHomeButton 1008//((Left Origin = 119, Top Origin = 0,
Width = 13, Height = 13, Usable = 1, Anchor Left = 1, Anchor Right = 0,
Standard)
#define MainOpenButton 1011//((Left Origin = 93, Top Origin = 0,
Width = 13, Height = 13, Usable = 1, Anchor Left = 1, Anchor Right = 0,
Standard)
#define MainLeftArrowBitmap 1000//((Left Origin = 107, Top Origin = 0,
Bitmap Resource ID = 1000, Usable = 1)
#define MainRightArrowBitmap 1100//((Left Origin = 134, Top Origin = 0,
Bitmap Resource ID = 1100, Usable = 1)
#define MainHomeBitmap 1200//((Left Origin = 119, Top Origin = 0,
Bitmap Resource ID = 1200, Usable = 1)
#define MainOpenBitmap 2711//((Left Origin = 93, Top Origin = 0,
Bitmap Resource ID = 2711, Usable = 1)
#define MainViewWidget 1002//((Left Origin = 0, Top Origin = 18,
Width = 133, Height = 142, Usable = 1)
#define MainBookmarkPopListList 1009//((Left Origin = 86, Top Origin = 1,
Width = 72, Usable = 0, Font = Standard, Visible Items = 0)
#define MainBookmarkPopTrigger 1010//((Left Origin = 144, Top Origin = 0,
Width = 0, Height = 13, Usable = 1, Anchor Left = 1, Anchor Right = 1009)
#define MainVerticalScrollbar 1001//((Left Origin = 153, Top Origin = 17,
Width = 7, Height = 143, Usable = 1, Value = 0, Maximum Value = 0, Page
Size = 0)
#define MainHorizontalScrollbar 1013//((Left Origin = 0, Top Origin = 153,
Width = 151, Height = 7, Usable = 1, Value = 0, Minimum Value = 0, Page
Size = 0)

// Resource: tFRM 1100
#define TextEntryForm 1100//((Left Origin = 2, Top Origin = 13,
Width = 156, Height = 144, Usable = 1, Modal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID
= 0, Default Button ID = 0)
#define TextEntryOKButton 1103//((Left Origin = 34, Top Origin = 129,
Width = 36, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Font =
Standard)
#define TextEntryCancelButton 1104//((Left Origin = 87, Top Origin = 129,
Width = 36, Height = 12, Usable = 1, Anchor Left = 1, Anchor Right = 1, Font =
Standard)
#define TextEntryTextField 1101//((Left Origin = 3, Top Origin = 20,
Width = 144, Height = 104, Usable = 1, Editable = 1, Underline = 1, Single Line = 0, Dynamic
Size = 0, Left Justified = 1, Max Characters = 80, Font = Standard, Auto Shift = 0, Has
Scroll Bar = 0, Numeric = 0)
#define TextEntryTextScrollbar 1102//((Left Origin = 146, Top Origin = 20,
Width = 7, Height = 105, Usable = 0, Value = 0, Minimum Value = 0, Maximum Value = 0, Page
Size = 0)

// Resource: Talt 2570
#define BookmarkDetailsAlert 2570
#define BookmarkDetailsOK 0

// Resource: Talt 2571
#define BookmarkDeleteAlert 2571
#define BookmarkDeleteOK 0

// Resource: Talt 1000
#define DownloadErrorAlert 1000
#define DownloadErrorOK 0

// Resource: Talt 1100
#define CustomAlert 1100
#define CustomOK 0

// Resource: MBAR 2640
#define BrowsingFormMenuBar 2640

// Resource: MBAR 2645
#define BookmarkFormMenuBar 2645

// Resource: MBAR 2650
#define BookmarkEditFormMenuBar 2650

```

```

// Resource: MBAR 2651
#define GotoFormMenuBar 2651

// Resource: MENU 2680
#define OptionsMenu 2680
#define OptionsPreferences 2680
#define OptionsAboutBlazer 2682

// Resource: MENU 2650
#define BrowserPageMenu 2650
#define BrowserPageGoto 2650
#define BrowserPageBack 2652
#define BrowserPageForward 2653
#define BrowserPageHome 2654
#define BrowserPageStop 2656

// Resource: MENU 2660
#define BookmarkEditMenu 2660
#define BookmarkEditCut 2660
#define BookmarkEditCopy 2661
#define BookmarkEditPaste 2662
#define BookmarkEditUndo 2663
#define BookmarkEditSelectAll 2664
#define BookmarkEditKeyboard 2665
#define BookmarkEditTraffici 2667

// Resource: MENU 2670
#define BrowserEditMenu 2670
#define BrowserEditBookmarks 2670

// Resource: tSTR 2450
#define DefaultProtocolString 2450
// "http://"

// Resource: tSTR 2460
#define DefaultHomePageString 2460
// "http://www.yahoo.com/"

// Resource: PICT 2710
#define VoyagerLogoBitmap 2710

// Resource: PICT 1000
#define LeftArrowBitmap 1000

// Resource: PICT 1100
#define RightArrowBitmap 1100

// Resource: PICT 1200
#define HomeBitmap 1200

// Resource: PICT 2711
#define OpenBitmap 2711

// Resource: PICT 1300
#define DownArrowBitmap 1300

#include "HTTPRequest.h"
#include "BLUtils.h"
#include <System/unix/sys_socket.h>
#include <System/unix/sys_errno.h>
#include <unix_string.h>

const int kBufferSize = 512;

// global:
Err errno;

// !!! make safe with buffer overflows

HTTPRequest::HTTPRequest( int sock,
char *hostname)
{
    mSocket = sock;
    mHostname = BLUtils::cloneString(hostname);
    mContentType = NULL;
    mContentLength = 0;
    mContentLocation = NULL;
    mEncodingType = kRaw;
}

```

```

HTTPRequest::~HTTPRequest()
{
    close(mSocket);
    delete[] mHostname;
    delete[] mContentType;
    delete[] mContentLocation;
}

int
HTTPRequest::requestFile(char *path)
{
    // swiped from the WebCrawler example
    if(path == NULL)
        return -1;

    int commandsSize =
        (int) StrLen(path)
        + (int) StrLen(mHostname)
        + 20 // Extra
        + 1 // NUL byte
        + 16; // Protocol filler...

    char buffer[kBufferSize];

    sprintf(buffer, "GET %s HTTP/1.1\r\n", path);

    // Send the GET command to the connected server.
    if(send(mSocket, buffer, StrLen(buffer), 0) > 0)
    {
        sprintf(buffer, "User-agent: Blazer\r\n");
        // IMP: The length of the command has to be sent!
        int retval = send(mSocket, buffer, StrLen(buffer), 0);

        if (retval <= 0)
            return -1;

        sprintf(buffer, "Host: %s\r\n\r\n", mHostname);
        // IMP: The length of the command has to be sent!
        retval = send(mSocket, buffer, StrLen(buffer), 0);

        if (retval <= 0)
            return -1;
        else
            return retval;
    }
    else
        return -1;
}

bool
HTTPRequest::parseHTTPHeader()
{
    int status = readStatusLine();
    int contentLength = 0;

    switch(status)
    {
        case 200:
            if(parseHeaders() == -1)
                return false;
            else
                return true;
        break;
        default:
            return false;
    }
}

```

```

break;
}

int
HTTPRequest::readStatusLine()
{
    char linebuff[kBufferSize];

    int majorVersion = 0;
    int minorVersion = 0;
    int status = 0;
    int length = 0;
    int pos = 0;

    if((length = readline(linebuff, kBufferSize)) == -1)
        return -1;
    if(length < 5)
        return -1;
    if(!strcmp(linebuff, "HTTP/", 5) == 0)
        return -1;
    pos = 5;
    if(pos >= length)
        return -1;
    while(linebuff[pos] != ' ')
    {
        pos++;
        if(pos >= length)
            return -1;
    }
    pos++;
    if((length - pos) > 0)
        status = (int) StrAtoi(&linebuff[pos]);
    else
        return -1;
    return status;
}

int
HTTPRequest::parseHeaders()
{
    char buf[kBufferSize];

    CharPtr originalContentLoc = mContentLocation;
    bool hasNewLocation = false;
    mEncodingType = kRaw;
    while(true)
    {
        if (readline(buf, kBufferSize) == -1)
            return -1;
        if (buf[0] == '\0')
            break;
        char *cp = strchr(buf, ':');
        if (cp == NULL)
            return -1;
        *cp = '\0';
        while (*(++cp) == ' ')
            ; //skip spaces
    }
}

```

```

if(StrCaseLessCompare(buf, "content-type") == 0)
{
    mContentType = BUUtils::cloneString(cp);
}
else if(StrCaseLessCompare(buf, "content-length") == 0)
{
    mContentLength = (int) StrAtoi(cp);
}
else if(StrCaseLessCompare(buf, "content-location") == 0)
{
    hasNewLocation = true;
    mContentLocation = BUUtils::cloneString(cp);
}
else if(StrCaseLessCompare(buf, "transfer-encoding") == 0)
{
    if(StrCaseLessCompare(cp, "chunked") == 0)
        mEncodingType = kChunked;
    else
        mEncodingType = kUnknown;
}
}

if(hasNewLocation)
{
    delete() originalContentLoc;
}
return 0;
}

int
HTTPRequest::readline(char *buf, int size)
{
    char c = 0;
    int rc = 0;
    int i;

    for (i=0; i<size-1; i++)
    {
        if ((rc = read(mSocket, &c, 1)) == 1)
        {
            if (c == '\n')
                break;
            if (c == '\r') {
                i--;
                continue;
            }
            buf[i] = c;
        }
        else if (rc == 0)
        {
            return -1;
        }
        else
        {
            if (errno == EWOULDBLOCK)
                rc++;
        }
    }

    buf[i] = '\0';
    return (int) StrLen(buf);
}

// HTTPRequest.h
#pragma once
class HTTPRequest
{
public:
    enum EncodingType { kUnknown=0, kChunked, kRaw };

```

```

int socket,
char *hostname);

~HTTPRequest();

int requestFile(char *path);

bool parseHTTPHeader();
int getContentLength();
char* getContentType();
char* getContentLocation();
EncodingType getEncodingType();

int getSocket() { return mSocket; }

protected:
    int readStatusLine();
    int parseHeaders();
    int readline(char *buf, int size);

    int mSocket;
    CharPtr mHostname;
    CharPtr mContentType;
    CharPtr mContentLocation;
    int mContentLength;
    EncodingType mEncodingType;
};

// Main.cpp
#include "BrowserApp.h"

static Err romVersionCompatible(DWord requiredVersion, Word launchFlags);

DWord PilotMain(Word cmd, Ptr /*cmdPBP*/, Word launchFlags)
{
    switch(cmd)
    {
        case sysAppLaunchCmdNormalLaunch:
        {
            Err error = romVersionCompatible(kMinOSVersionNeeded, launchFlags);
            if (error)
                return error;

            BrowserApp app;
            return app.launchNormal();
        }
        default:
            break;
    }
    return 0;
}

Err romVersionCompatible(DWord requiredVersion, Word launchFlags)
{
    DWord romVersion;

    // See if we're on in minimum required version of the ROM or later.
    FrGet(sysFrCreator, sysFrNumROMVersion, &romVersion);
    if (romVersion < requiredVersion)
    {
        if ((launchFlags & (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp)) ==
            (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp))
        {
            FrmAlert (RomIncompatibleAlert);
        }
        // Pilot 1.0 will continuously relaunch this app unless we switch to

```

```

// another safe one.
if (fromVersion < sysMakeROMVersion(2,0,0,sysROMStageRelease,0))
    AppLaunchWithCommand(sysFileCDefaultApp, sysAppLaunchCmdNormalLaunch, NULL);
}
return sysErrRomIncompatible;
}

return 0;
} // PrecompiledHeaders.pch++

#ifdef PILOT_PRECOMPILED_HEADERS_OFF
#include <SysAll.h>
#include <UIAll.h>

#define ENABLE_NETWORKING
#define ENABLE_NETWORKING

enum {
    firstNetLibEvent = 0x1000,
    firstWebLibEvent = 0x1100
};

#include <NetMgr.h>
#include <InetMgr.h>
#include <CTP.h>

#define BERKELEY_SOCKETS
#endif

#include "BrowserRaci_res.h"

const DWord kMinOSVersionNeeded = sysMakeROMVersion(3,0,0,sysROMStageRelease,0);

#define kHTTPGateway "proxy.blueark.net"
#define kHTTPGateway "haze.blueark.net"
#define kHTTPGateway "red.blueark.net"
#define kHTTPGateway "63.203.230.69"
#define kDefaultPage "blue.blueark.net/home.wml"
#define kDefaultPage "blue.blueark.net/~jkleid/options.wml"

#define kURLPrefix "/http/"

const DateType kExpirationDate = {96, 6, 18};
const int kHTTPPort = 80;
//const int kHTTPPort = 7777;

#pragma precompile_target "WMLBrowser.h++.mch"

#endif // WEXMLConstants.h

#pragma once

enum XMLGlobalToken
{
    TOK_SWITCH_PAGE = 0x0,
    TOK_END,
    TOK_ENTITY,
    TOK_STR_1,
    TOK_LITERAL,
    TOK_EXT_1_0 = 0x40,
    TOK_EXT_1_1,
    TOK_EXT_1_2,
    TOK_PI_1,
    TOK_LITERAL_C,
    TOK_EXT_T_0 = 0x80,
    TOK_EXT_T_1,
    TOK_EXT_T_2,
    TOK_STR_T,
}

```

```

TOK_LITERAL_A,
TOK_EXT_0 = 0x80,
TOK_EXT_1,
TOK_EXT_2,
TOK_OPAQUE,
TOK_LITERAL_AC
};

enum WMLTagToken
{
    TAG_A = 0x1C,
    TAG_TD,
    TAG_TR,
    TAG_TABLE,
    TAG_P = 0x20,
    TAG_POSTFIELD,
    TAG_ANCHOR,
    TAG_ACCESS,
    TAG_B,
    TAG_BIG,
    TAG_BR,
    TAG_CARD,
    TAG_DO,
    TAG_EM,
    TAG_FIELDSET,
    TAG_GO,
    TAG_HEAD,
    TAG_I,
    TAG_IMG,
    TAG_INPUT,
    TAG_META = 0x30,
    TAG_NOOP,
    TAG_PREV,
    TAG_ONEVENT,
    TAG_OPGROUP,
    TAG_OPTION,
    TAG_REFRESH,
    TAG_SELECT,
    TAG_SMALL,
    TAG_STRONG,
    // NULL
    TAG_TEMPLATE = 0x3B,
    TAG_TIMER,
    TAG_U,
    TAG_SETVAR,
    TAG_WML
};

enum WMLAttr
{
    ATTR_ACCEPT_CHARSET = 0x05,
    ATTR_ALIGN_BOTTOM,
    ATTR_ALIGN_CENTER,
    ATTR_ALIGN_LEFT,
    ATTR_ALIGN_MIDDLE,
    ATTR_ALIGN_RIGHT,
    ATTR_ALIGN_TOP,
    ATTR_ALT,
    ATTR_CONTENT,
    // NULL
    ATTR_DOMAIN = 0x0F,
    ATTR_EMPTYTOK_FALSE = 0x10,
    ATTR_EMPTYTOK_TRUE,
    ATTR_FORMAT,
    ATTR_HEIGHT,
    ATTR_HSPACE,
    ATTR_IVALUE,
    ATTR_INAME,
    // NULL
}

```



```

ATTR_LABEL = 0x18,
ATTR_LOCALSRC,
ATTR_MAXLENGTH,
ATTR_METHOD_GET,
ATTR_METHOD_POST,
ATTR_MODE_NOWRAP,
ATTR_MULTIPLE_FALSE,
ATTR_MULTIPLE_TRUE = 0x20,
ATTR_NAME,
ATTR_NEWCONTEXT_FALSE,
ATTR_NEWCONTEXT_TRUE,
ATTR_ONPICK,
ATTR_ONENTERBACKWARD,
ATTR_ONENTERFORWARD,
ATTR_ONTIMER,
ATTR_OPTIONAL_FALSE,
ATTR_OPTIONAL_TRUE,
ATTR_PATH,
// NULL
// NULL
ATTR_SCHEME = 0x2E,
ATTR_SENDFERFER_FALSE,
ATTR_SENDFERFER_TRUE = 0x30,
ATTR_SIZE,
ATTR_SRC,
ATTR_ORDERED_TRUE,
ATTR_ORDERED_FALSE,
ATTR_TABINDEX,
ATTR_TITLE,
ATTR_TYPE,
ATTR_TYPE_ACCEPT,
ATTR_TYPE_DELETE,
ATTR_TYPE_HELP,
ATTR_TYPE_PASSWORD,
ATTR_TYPE_ONPICK,
ATTR_TYPE_ONENTERBACKWARD,
ATTR_TYPE_ONENTERFORWARD,
ATTR_TYPE_ONTIMER,
// NULL
// NULL
// NULL
// NULL
ATTR_TYPE_OPTIONS = 0x45,
ATTR_TYPE_PREV,
ATTR_TYPE_RESET,
ATTR_TYPE_TEXT,
ATTR_TYPE_VND,
ATTR_HREF,
ATTR_HREF_HTTP,
ATTR_HREF_HTTPS,
ATTR_VALUE,
ATTR_VSPACE,
ATTR_WIDTH,
ATTR_XML_LANG = 0x50,
// NULL
ATTR_ALIGN = 0x52,
ATTR_COLUMNS,
ATTR_CLASS,
ATTR_ID,
ATTR_FORUM_FALSE,
ATTR_FORUM_TRUE,
ATTR_SRC_HTTP,
ATTR_SRC_HTTPS,
ATTR_HTTP_EQUIV,
ATTR_HTTP_EQUIV_CONTENT_TYPE,
ATTR_CONTENT_APPLICATION_VND_WAP_WMLC_CHARSET,
ATTR_HTTP_EQUIV_EXPIRES,
ATTR_ACCESS_KEY,

```

```

ATTR_ENTTYPE,
ATTR_ENTTYPE_URL_ENCODED = 0x60,
ATTR_ENTTYPE_FORM_DATA
};

enum WMLValue
{
    VALUE_DOT_COM = 0x85,
    VALUE_DOT_EDU,
    VALUE_DOT_NET,
    VALUE_DOT_ORG,
    VALUE_ACCEPT,
    VALUE_BOTTOM,
    VALUE_CLEAR,
    VALUE_DELETE,
    VALUE_HELP,
    VALUE_HTTP,
    VALUE_HTTP_WWW,
    VALUE_HTTPS = 0x90,
    VALUE_HTTPS_WWW,
    // NULL
    VALUE_MIDDLE = 0x93,
    VALUE_NOWRAP,
    VALUE_ONPICK,
    VALUE_ONENTERBACKWARD,
    VALUE_ONENTERFORWARD,
    VALUE_OPTIONS,
    VALUE_PASSWORD,
    VALUE_RESET,
    // NULL
    VALUE_TEXT = 0x9D,
    VALUE_TOP,
    VALUE_UNKNOWN,
    VALUE_WRAP = 0xA0,
    VALUE_WWW
};

// WEXMLParser.cpp
#include "WEXMLParser.h"
#include "WEXMLConstants.h"
#include "BLWMLTable.h"
#include "BLWMLActions.h"
#include "BLUtils.h"

BLGlyph *
WEXMLParser::sCurrentGlyph = NULL;

int
WEXMLParser::sOpenTables = 0;

const int kInitStackSize = 8;

const CharPtr kTypeStrings[] =
{
    "Unknown Action",
    "Accept",
    "Previous",
    "Help",
    "Reset",
    "Options",
    "Delete"
};

const TagHandlerType cTags[] =
{
    &WEXMLParser::tag_a,
    &WEXMLParser::tag_td,
    &WEXMLParser::tag_tr,
    &WEXMLParser::tag_table,

```

```

    else
    {
        mStringTable = NULL;
    }
}

ULONG
WBXMLParser::readData(CharPtr buffer, ULONG length)
{
    try {
        mLength = length;
        mBuffer = buffer;
        if (!mHeaderDone)
        {
            ULONG workingPos = mPos;
            parseHeader(workingPos);
            mHeaderDone = true;
            mPos = workingPos;
            TopItem * top = new TopItem;
            mGlyphStack.add(top);
            mStateStack.add(mCurrentTextState);
        }
        if (mCurrTextBuffer && !text_run(mPos))
        {
            setCurrGlyph();
            return mPos;
        }
        sCurrentGlyph = NULL;
        while (mPos < length)
        {
            ULONG workingPos = mPos;
            if (workingPos >= mLength)
                break;
            mCurrentToken = (Byte) mBuffer[workingPos++];
            Byte tag = (Byte) (mCurrentToken & 0x3F);
            if ((tag >= TAG_A) && (tag <= TAG_WML))
            {
                (this->cTags[tag - TAG_A])(workingPos);
            }
            else
            {
                switch(tag)
                {
                    case TOK_END:
                        BGLlyph * popped = mGlyphStack.popLastElement();
                        mStateStack.removeLast();
                        mCurrentTextState = mStateStack.getLastElement();
                        if (dynamic_cast<WMLTable *>(popped) != NULL)
                            sOpenTables--;
                        else if (dynamic_cast<WMLCard *>(popped) != NULL)
                            dynamic_cast<WMLCard *>(popped)->doneParsing();
                        else if (dynamic_cast<WMLRoot *>(popped) != NULL)
                            return 0;
                        else if (dynamic_cast<TopItem *>(popped) != NULL)
                            throw
                                FatalParsingError(FatalParsingError::UnexpectedEndOfStack);
                        break;
                }
            }
        }
    }
}

```

```

WBXMLParser::tag_P,
WBXMLParser::tag_postfield,
WBXMLParser::tag_anchor,
WBXMLParser::tag_access,
WBXMLParser::tag_strong,
WBXMLParser::tag_big,
WBXMLParser::tag_br,
WBXMLParser::tag_card,
WBXMLParser::tag_do,
WBXMLParser::tag_em,
WBXMLParser::tag_fieldset,
WBXMLParser::tag_go,
WBXMLParser::tag_head,
WBXMLParser::tag_img,
WBXMLParser::tag_input,
WBXMLParser::tag_meta,
WBXMLParser::tag_noop,
WBXMLParser::tag_prev,
WBXMLParser::tag_onevent,
WBXMLParser::tag_optgroup,
WBXMLParser::tag_option,
WBXMLParser::tag_refresh,
WBXMLParser::tag_select,
WBXMLParser::tag_small,
WBXMLParser::tag_strong,
WBXMLParser::tag_unknown_tag,
WBXMLParser::tag_template,
WBXMLParser::tag_timer,
WBXMLParser::tag_u,
WBXMLParser::tag_setvar,
WBXMLParser::tag_wml
};

```

```

WBXMLParser::WBXMLParser()
: mGlyphStack(kInitStackSize),
  mStateStack(kInitStackSize)
{
    mStringTable = NULL;
    mHeaderDone = false;
    mPos = 0;
    mRoot = NULL;
    mCurrTextBuffer = NULL;
    mCurrTextBufferLen = 0;
    mCurrSelectP = NULL;
}

```

```

WBXMLParser::~WBXMLParser()
{
}

```

```

void
WBXMLParser::parseHeader(ULONG spos)
{
    UChar versionNum = (UChar) mBuffer[pos++];
    if ((versionNum < 0x1) || (versionNum > 0x2))
        throw FatalParsingError(FatalParsingError::kFileTypeError);
    mPublicId = parseMultiByteInt(pos);
    if (mPublicId == 0)
        mID_Index = (Byte) mBuffer[pos++];
    mCharSet = parseMultiByteInt(pos);
    mStringTableLen = parseMultiByteInt(pos);
    if (mStringTableLen > 0)
    {
        mStringTable = mBuffer[pos];
        pos += mStringTableLen;
    }
}

```

```

        case TOK_STR_I:
        case TOK_STR_T:
            workingPos--;
            if(!text_run(workingPos))
            {
                setCurrGlyph();
                return mPos;
            }
            break;
        )
    )
    mPos = workingPos;
    )
    catch(UnexpectedEOF)
    {
        // need more bytes to continue
    }
    catch(FatalParsingError)
    {
        // serious error, cannot continue
        return 0;
    }
    if((dynamic_cast<TopItem*>(mGlyphStack.getLastElement()) != NULL) && (mRoot != NULL))
    {
        sCurrentGlyph = NULL;
        return 0;
    }
    else
    {
        setCurrGlyph();
        if(mPos == 0)
            return 1;
        return mPos;
    }
    )
    )

void
WBXMLParser::setCurrGlyph()
{
    if((sCurrentGlyph == NULL) && (mGlyphStack.getNumElements() >= 1))
    {
        sCurrentGlyph = mGlyphStack.getLastElement();
    }
    )

ULong
WBXMLParser::parseMultiByteInt(ULong &pos)
{
    // !!! untested for big ints, as far as I know
    ULong result = 0;
    while(pos < mLength)
    {
        UInt digit = mBuffer[pos++];
        if((digit & 0x80) == 0)
        {
            // no more digits
            result |= digit;
            return result;
        }
        digit &= (UInt) ~0x80;
        result |= digit;
        result <<= 8;
    }
    throw UnexpectedEOF();
}

Byte
WBXMLParser::parseByte(ULong &pos)
{
    if(pos < mLength)
    {
        return (Byte) mBuffer[pos++];
    }
    throw UnexpectedEOF();
}

CharPtr
WBXMLParser::parseString(ULong &pos, ULong * lengthP, bool parseType)
{
    CharPtr string;
    Byte token = parseByte(pos);
    if(!parseType || (token == TOK_STR_I))
    {
        ULong length = stringLength(pos);
        string = &mBuffer[pos];
        pos += length + 1;
        if(lengthP)
            *lengthP = length;
    }
    else if(token == TOK_STR_T)
    {
        ULong offset = parseMultiByteInt(pos);
        string = mStringTable + offset;
        if(lengthP)
            *lengthP = Strlen(string); // !!! not safe
    }
    return string;
}

CharPtr
WBXMLParser::parseURL(ULong &pos)
{
    Byte token;
    CharPtr string;
    BLVector<CharPtr, 4> stringVector;
    ULong totalLength = 0;
    do {
        string = NULL;
        ULong strLength;
        token = parseByte(pos);
        switch(token)
        {
            case TOK_STR_I:
            case TOK_STR_T:
                string = parseString(--pos, &strLength);
                stringVector.add(string);
                totalLength += strLength;
                string = NULL;
                break;
            case ATTR_SRC:
            case ATTR_HREF:
                break;
            case ATTR_HREF_HTTP:
            case ATTR_SRC_HTTP:
                string = "http://";
                break;
        }
    }
}

```

```

case ATTR_HREF_HTTPS:
case ATTR_SRC_HTTPS:
    string = "https://";
    break;

case VALUE_HTTP:
    string = "http://";
    break;

case VALUE_HTTPS:
    string = "https://";
    break;

case VALUE_HTTP_WWW:
    string = "http://www.";
    break;

case VALUE_HTTPS_WWW:
    string = "https://www.";
    break;

case VALUE_WWW:
    string = "www.";
    break;

case VALUE_DOT_COM:
    string = ".com/";
    break;

case VALUE_DOT_ORG:
    string = ".org/";
    break;

case VALUE_DOT_NET:
    string = ".net/";
    break;

case VALUE_DOT_EDU:
    string = ".edu/";
    break;

case TOK_END:
default:
    pos--;
    CharPtr urlP = NULL;
    ULONG lenUsed = 0;
    if (totalLength == 0)
        return NULL;
    if (stringVector.getNumElements() == 1)
        return stringVector.getElementAt(0);
    urlP = (CharPtr) BLGlyph::newChunk(totalLength + 1);
    BLIterator<CharPtr, 4> iter(stringVector);
    while (iter.hasNext())
    {
        CharPtr piece = iter.getNext();
        StrCopy(urlP + lenUsed, piece);
        lenUsed += StrLen(piece);
    }
    return urlP;
}

if (string != NULL)
{
    stringVector.add(string);
    totalLength += StrLen(string);
} while (pos < mLength);
}

ULONG
WBXMLParser::stringLength(ULONG &pos)
{
    ULONG length = 0;
    while (mBuffer[pos + length])
    {
        length++;
        if ((pos + length) >= mLength)
            throw UnexpectedEOF(pos + length, &mBuffer[pos]);
    }
    return length;
}

WMLTagToken
WBXMLParser::getEnclosingContext(BLGlyph * &foundGlyph)
{
    int i = mGlyphStack.getNumElements();
    while (i > 0)
    {
        foundGlyph = mGlyphStack.getElementAt(--i);
        if (dynamic_cast<WMLParagraph*>(&foundGlyph))
            return TAG_P;
        else if (dynamic_cast<WMLCard*>(&foundGlyph))
            return TAG_CARD;
        else if (dynamic_cast<WMLHead*>(&foundGlyph))
            return TAG_HEAD;
        else if (dynamic_cast<WMLTemplate*>(&foundGlyph))
            return TAG_TEMPLATE;
        else if (dynamic_cast<WMLRoot*>(&foundGlyph))
            return TAG_WML;
    }
    throw FatalParsingError(FatalParsingError::kRootUndefined);
}

bool
WBXMLParser::text_run(ULONG &pos)
{
    ULONG length;
    CharPtr string;
    bool done = false;
    try
    {
        string = parseString(pos, &length, mCurrTextBufferLen == 0);
        done = true;
    }
    catch (UnexpectedEOF eof)
    {
        length = eof.mPos - pos;
        mPos = pos = eof.mPos - 1;
        if (mCurrTextBuffer == NULL)
            mCurrTextBuffer = (CharPtr) eof.mData;
    }
}

```

```

        mCurrTextBufferLen += length;
    }
    return false;
}

if((mCurrTextBufferLen > 0) && (mCurrTextBuffer != NULL))
{
    string = mCurrTextBuffer;
    length += mCurrTextBufferLen;
}

if(length > 0)
{
    new WMLTextRun( mGlyphStack.getLastElement(),
        string,
        (int) length,
        mCurrTextState);
}

mCurrTextBuffer = NULL;
mCurrTextBufferLen = 0;
return true;
}

void
WBXMLParser::unknown_tag(ULONG &pos)
{
    if(hasAttribute(mCurrToken))
    {
        Byte token;

        do {
            token = parseByte(pos);
        } while(token != TOK_END);
    }

    if(!hasContent(mCurrToken))
        return;

    mGlyphStack.add(mGlyphStack.getLastElement());
    mStateStack.add(mCurrTextState);
}

void
WBXMLParser::tag_a(ULONG &pos)
{
    WMLHyperlink link;

    if(hasAttribute(mCurrToken))
    {
        Byte token;

        do {
            token = parseByte(pos);
        } while(token != TOK_END);

        switch(token)
        {
            case ATTR_HREF:
            case ATTR_HREF_HTTP:
            case ATTR_HREF_HTTPS:
                link.mHref = parseURL(--pos);
                break;
        }
    }

    while(token != TOK_END);

    if(!hasContent(mCurrToken))
        return;

    WMLHyperlink * linkP = new WMLHyperlink(link);
    mCurrTextState.action = linkP;
}

```

```

        mCurrTextState.underline = grayUnderline;
    }
    mGlyphStack.add(mGlyphStack.getLastElement());
    mStateStack.add(mCurrTextState);
}

void
WBXMLParser::tag_td(ULONG &pos)
{
    if(!hasContent(mCurrToken))
        return;

    WMLTableCell * cell = new WMLTableCell(mGlyphStack.getLastElement());

    mGlyphStack.add((WMLParagraph*) cell);
    mStateStack.add(mCurrTextState);
}

void
WBXMLParser::tag_tr(ULONG &pos)
{
    if(!hasContent(mCurrToken))
        return;

    WMLTableRow * row = new WMLTableRow(mGlyphStack.getLastElement());

    mGlyphStack.add(row);
    mStateStack.add(mCurrTextState);
}

void
WBXMLParser::tag_table(ULONG &pos)
{
    WMLTable table;

    if(hasAttribute(mCurrToken))
    {
        Byte token;

        do {
            token = parseByte(pos);
        } while(token != TOK_END);

        switch(token)
        {
            case ATTR_COLUMNS:
                table.mNumColumns = (int) StrAtoi(parseString(pos));
                break;
            case ATTR_ALIGN:
                table.mAlignArray = parseString(pos);
                break;
            case ATTR_TITLE:
                table.mTitle = parseString(pos);
                break;
        }
    }

    while(token != TOK_END);

    if(!hasContent(mCurrToken))
        return;

    sOpenTables++;

    WMLTable * tableP = new WMLTable(mGlyphStack.getLastElement(), table);

    mGlyphStack.add(tableP);
    mStateStack.add(mCurrTextState);
}

void

```

```

WBXMLParser::tag_p(ULONG &pos)
{
    WMLParagraph paragraph;
    if (hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_MODE_WRAP:
                    paragraph.bits.misWrapOn = true;
                    break;
                case ATTR_MODE_NOWRAP:
                    paragraph.bits.misWrapOn = false;
                    break;
                case ATTR_ALIGN_LEFT:
                    paragraph.mAlign = BLGlyph::kLeft;
                    break;
                case ATTR_ALIGN_RIGHT:
                    paragraph.mAlign = BLGlyph::kRight;
                    break;
                case ATTR_ALIGN_CENTER:
                    paragraph.mAlign = BLGlyph::kCenter;
                    break;
            }
        } while(token != TOK_END);
    }
    if (!hasContent(mCurrentToken))
        return;
    WMLParagraph * paragraphp = new WMLParagraph(mGlyphStack.getLastElement(), paragraph);
    mGlyphStack.add(paragraphp);
    mCurrentTextState.bits.wordWrap = paragraph.bits.misWrapOn;
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_postfield(ULONG &pos)
{
    WMLPostfield postfield;
    if (hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_NAME:
                    postfield.mName = parseString(pos);
                    break;
                case ATTR_VALUE:
                    postfield.mValue = parseString(pos);
                    break;
            }
        } while(token != TOK_END);
    }
    if (!hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedContent);
}

```

```

WMLPostfield * postfieldp = new WMLPostfield(mGlyphStack.getLastElement(), postfield);
WBXMLParser::tag_anchor(ULONG &pos)
{
    WMLAnchor anchor;
    if (hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_TITLE:
                    anchor.mTitle = parseString(pos);
                    break;
                case ATTR_ACCESS_KEY:
                    anchor.mAccessKey = parseString(pos);
                    break;
            }
        } while(token != TOK_END);
    }
    if (!hasContent(mCurrentToken))
        return;
    WMLAnchor * anchorp = new WMLAnchor(mGlyphStack.getLastElement(), anchor);
    mCurrentTextState.action = anchorp;
    mCurrentTextState.underline = grayUnderline;
    mGlyphStack.add(anchorp);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_access(ULONG &pos)
{
    if (mRoot == NULL)
        throw FatalParsingError(FatalParsingError::kRootUndefined);
    if (hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_DOMAIN:
                    mRoot->mDomainAccess = parseString(pos);
                    break;
                case ATTR_PATH:
                    mRoot->mPathAccess = parseString(pos);
                    break;
            }
        } while(token != TOK_END);
    }
    if (!hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedContent);
}

void
WBXMLParser::tag_big(ULONG &pos)

```



```

        if (hasAttribute(mCurrentToken))
        {
            throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
        }
        mGlyphStack.add(mGlyphStack.getLastElement());
        mStateStack.add(mCurrentTextState);
    }

    void
    WEXMLParser::tag_fieldset(ULONG &pos)
    {
        // page 52

        WMLFieldSet fieldset;

        if (hasAttribute(mCurrentToken))
        {
            Byte token;

            do {
                token = parseByte(pos);
                switch(token)
                {
                    case ATTR_TITLE:
                        fieldset.mTitle = parseString(pos);
                        break;
                }
            } while(token != TOK_END);
        }

        WMLFieldSet * fieldsetP = new WMLFieldSet(mGlyphStack.getLastElement(), fieldset);
    }

    void
    WEXMLParser::tag_go(ULONG &pos)
    {
        WMLGo go;

        if (hasAttribute(mCurrentToken))
        {
            Byte token;

            do {
                token = parseByte(pos);
                switch(token)
                {
                    case ATTR_HREF:
                    case ATTR_HREF_HTTP:
                    case ATTR_HREF_HTTPS:
                        go.mHref = parseURL(--pos);
                        break;
                    case ATTR_SENDREFERER_FALSE:
                        go.mSendReferer = false;
                        break;
                    case ATTR_SENDREFERER_TRUE:
                        go.mSendReferer = true;
                        break;
                    case ATTR_METHOD_GET:
                        go.mMethod = WMLGo::kGet;
                        break;
                    case ATTR_METHOD_POST:
                        go.mMethod = WMLGo::kPost;
                        break;
                    case ATTR_ACCEPT_CHARSET:
                        go.mEncodingType = parseString(pos);
                        break;
                }
            }
        }
    }

```



```

    } while(token != TOK_END);

    WMLGo * goP = new WMLGo(mGlyphStack.getLastElement(), go);

    if(mCurrentTextState.action->setAction(goP);
        mCurrentTextState.action)
    {
        return;
    }

    mGlyphStack.add(goP);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_head(ULONG &pos)
{
    if(hasAttribute(mCurrentToken))
    {
        Byte token;

        do {
            token = parseByte(pos);
        } while(token != TOK_END);
    }

    if(!hasContent(mCurrentToken))
        return;

    WMLHead * head = new WMLHead(mGlyphStack.getLastElement());

    mGlyphStack.add(head);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_img(ULONG &pos)
{
    WMLImage image(mCurrentTextState);

    if(hasAttribute(mCurrentToken))
    {
        Byte token;

        do {
            token = parseByte(pos);

            switch(token)
            {
                case ATTR_SRC:
                case ATTR_SRC_HTTP:
                case ATTR_SRC_HTTPS:
                    image.mSrc = parseURL(--pos);
                    break;

                case ATTR_HEIGHT:
                    parseString(pos); // ignore, for now
                    break;

                case ATTR_WIDTH:
                    parseString(pos); // ignore, for now
                    break;

                case ATTR_ALT:
                    image.mAltText = parseString(pos);
                    break;

                case ATTR_VSPACE:
                    image.mSpace.y = (Short) StrAtol(parseString(pos));
                    break;

                case ATTR_HSPACE:
                    image.mSpace.x = (Short) StrAtol(parseString(pos));
                    break;
            }
        } while(token != TOK_END);

        BLGlyph * foundGlyph;

        if(getEnclosingContext(foundGlyph) == TAG_P)
            image.mParentParagraph = dynamic_cast<WMLParagraph*>(foundGlyph);

        mImages.add(new WMLImage(mGlyphStack.getLastElement(), image));

        if(hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::KUnexpectedContent);
    }
}

void
WBXMLParser::tag_input(ULONG &pos)
{
    // page 50

    WMLInput input;

    if(hasAttribute(mCurrentToken))
    {
        Byte token;

        do {
            token = parseByte(pos);

            switch(token)
            {
                case ATTR_NAME:
                    input.mName = parseString(pos);
                    break;

                case ATTR_VALUE:
                    input.mValue = parseString(pos);
                    break;

                case ATTR_TITLE:
                    input.mTitle = parseString(pos);
                    break;

                case ATTR_ACCESS_KEY:
                    input.mAccessKey = parseString(pos);
                    break;

                case ATTR_FORMAT:
                    input.mFormat = parseString(pos);
                    break;
            }
        } while(token != TOK_END);

        BLGlyph * foundGlyph;

        if(getEnclosingContext(foundGlyph) == TAG_P)
            image.mParentParagraph = dynamic_cast<WMLParagraph*>(foundGlyph);

        mImages.add(new WMLImage(mGlyphStack.getLastElement(), image));

        if(hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::KUnexpectedContent);
    }
}

void
WBXMLParser::tag_input(ULONG &pos)
{
    // page 50

    WMLInput input;

    if(hasAttribute(mCurrentToken))
    {
        Byte token;

        do {
            token = parseByte(pos);

            switch(token)
            {
                case ATTR_NAME:
                    input.mName = parseString(pos);
                    break;

                case ATTR_VALUE:
                    input.mValue = parseString(pos);
                    break;

                case ATTR_TITLE:
                    input.mTitle = parseString(pos);
                    break;

                case ATTR_ACCESS_KEY:
                    input.mAccessKey = parseString(pos);
                    break;

                case ATTR_FORMAT:
                    input.mFormat = parseString(pos);
                    break;
            }
        } while(token != TOK_END);

        BLGlyph * foundGlyph;

        if(getEnclosingContext(foundGlyph) == TAG_P)
            image.mParentParagraph = dynamic_cast<WMLParagraph*>(foundGlyph);

        mImages.add(new WMLImage(mGlyphStack.getLastElement(), image));

        if(hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::KUnexpectedContent);
    }
}

```

```

        input.mFormat = parseString(pos);
        break;
    case ATTR_SIZE:
        input.mSize = (int) StrAToI(parseString(pos));
        break;
    case ATTR_MAXLENGTH:
        input.mMaxLength = (int) StrAToI(parseString(pos));
        break;
    case ATTR_TABINDEX:
        input.mTabIndex = (int) StrAToI(parseString(pos));
        break;
    case ATTR_TYPE_TEXT:
        input.mType = WMLInput::kText;
        break;
    case ATTR_TYPE_PASSWORD:
        input.mType = WMLInput::kPassword;
        break;
    case ATTR_EMPTYTOK_FALSE:
        input.mEmptytok = false;
        break;
    case ATTR_EMPTYTOK_TRUE:
        input.mEmptytok = true;
        break;
    } while(token != TOK_END);
}
if(hasContent(mCurrentToken))
    throw FatalParsingError(FatalParsingError::kUnexpectedContent);
WMLInput * inputP = new WMLInput(mGlyphStack.getLastElement(), input);
}

void
WBXMLParser::tag_meta(ULong &pos)
{
    if(hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_HTTP_EQUIV:
                case ATTR_HTTP_EQUIV_CONTENT_TYPE:
                case ATTR_HTTP_EQUIV_EXPIRES:
                    parseString(pos);
                    break;
                case ATTR_NAME:
                    parseString(pos);
                    break;
                case ATTR_FORUM_FALSE:
                case ATTR_FORUM_TRUE:
                    break;
                case ATTR_CONTENT:
                    parseString(pos);
                    break;
                case ATTR_SCHEME:
                    parseString(pos);
                    break;
            }
        }
    }
}

```

```

    } while(token != TOK_END);
    if(hasContent(mCurrentToken))
    {
        mGlyphStack.add(mGlyphStack.getLastElement());
        mStateStack.add(mCurrentTextState);
    }
}

void
WBXMLParser::tag_noop(ULong &pos)
{
    if(hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    if(hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedContent);
    if(mCurrentTextState.action)
        mCurrentTextState.action->setAction(NULL);
}

void
WBXMLParser::tag_prev(ULong &pos)
{
    if(hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    WMLPrevious * previous = new WMLPrevious(mGlyphStack.getLastElement());
    if(mCurrentTextState.action)
        mCurrentTextState.action->setAction(previous);
    if(!hasContent(mCurrentToken))
        return;
    mGlyphStack.add(previous);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_onevent(ULong &pos)
{
    WMLAttr attr = (WMLAttr) 0x0;
    if(hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_TYPE:
                    parseString(pos);
                    // unknown value
                    break;
                case ATTR_TYPE_ACCEPT:
                case ATTR_TYPE_DELETE:
                case ATTR_TYPE_HELP:
                case ATTR_TYPE_PASSWORD:
                case ATTR_TYPE_ONPICK:
                case ATTR_TYPE_ONENTERBACKWARD:
                case ATTR_TYPE_ONENTERFORWARD:
                case ATTR_TYPE_ONTIMER:
                case ATTR_ONENTERBACKWARD:
                case ATTR_ONENTERFORWARD:
                case ATTR_ONTIMER:
                case ATTR_ONPICK:
            }
        }
    }
}

```

```

        attr = (WMLAttr) token;
        break;
    }
    while(token != TOK_END);

    if(!hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kContentExpected);

    BGLymph * foundGlyph;
    WMLIntrinsicEvents * eventSP = NULL;

    switch(getEnclosingContext(foundGlyph))
    {
        case TAG_TEMPLATE:
            eventSP = &dynamic_cast<WMLTemplate*>(foundGlyph)->getIntrinsicEvents();
            break;

        case TAG_CARD:
            eventSP = &dynamic_cast<WMLCard*>(foundGlyph)->getIntrinsicEvents();
            break;

        case TAG_OPTION:
            // !!! Fill in later.
            default:
                break;
    }

    WMLOnEvent * onevent = new WMLOnEvent(mGlyphStack.getLastElement());

    if((attr != 0x0) && eventSP)
    {
        switch(attr)
        {
            case ATTR_ONENTERBACKWARD:
                case ATTR_TYPE_ONENTERBACKWARD:
                    eventSP->mOnEnterBackward = onevent;
                    break;

            case ATTR_ONENTERFORWARD:
                case ATTR_TYPE_ONENTERFORWARD:
                    eventSP->mOnEnterForward = onevent;
                    break;

            case ATTR_ONTIMER:
                case ATTR_TYPE_ONTIMER:
                    eventSP->mOnTimer = onevent;
                    break;

            case ATTR_ONPICK:
                case ATTR_TYPE_ONPICK:
                    eventSP->mOnPick = onevent;
                    break;
        }
    }

    mCurrentTextState.action = onevent;

    mGlyphStack.add(onevent);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_optgroup(ULONG tpos)
{
    // page 47
    WMLOptgroup optgroup;

    if(hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(tpos);
            switch(token)
            {
                case ATTR_TITLE:
                    option.mTitle = parseString(tpos);
                    break;

                case ATTR_VALUE:
                    option.mValue = parseString(tpos);
                    break;

                case ATTR_ONPICK:
                    option.mOnPick = parseURL(tpos);
                    break;
            }
        } while(token != TOK_END);

        if(!hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kContentExpected);

        mCurrSelectP->incrementOptionCount();

        WMLOption * optionP = new WMLOption(mGlyphStack.getLastElement(), option);

        mGlyphStack.add(optionP);
        mStateStack.add(mCurrentTextState);
    }
}

void
WBXMLParser::tag_refresh(ULONG tpos)
{
    // page 48
    WMLRefresh refresh;

    if(hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(tpos);
            switch(token)
            {
                case ATTR_REFRESH:
                    refresh.mRefresh = parseString(tpos);
                    break;
            }
        } while(token != TOK_END);

        if(!hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kContentExpected);

        mCurrRefreshP->incrementRefreshCount();

        WMLRefresh * refreshP = new WMLRefresh(mGlyphStack.getLastElement(), refresh);

        mGlyphStack.add(refreshP);
        mStateStack.add(mCurrentTextState);
    }
}

```

```

        token = parseByte(tpos);
        switch(token)
        {
            case ATTR_TITLE:
                optgroup.mTitle = parseString(tpos);
                break;

            case ATTR_VALUE:
                optgroup.mValue = parseString(tpos);
                break;

            case ATTR_ONPICK:
                optgroup.mOnPick = parseURL(tpos);
                break;
        }

        if(!hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kContentExpected);

        mCurrOptgroupP->incrementOptionCount();

        WMLOption * optionP = new WMLOption(mGlyphStack.getLastElement(), option);

        mGlyphStack.add(optionP);
        mStateStack.add(mCurrentTextState);
    }

    void
    WBXMLParser::tag_option(ULONG tpos)
    {
        // page 46
        WMLOption option(mCurrSelectP);

        if(hasAttribute(mCurrentToken))
        {
            Byte token;
            do {
                token = parseByte(tpos);
                switch(token)
                {
                    case ATTR_TITLE:
                        option.mTitle = parseString(tpos);
                        break;

                    case ATTR_VALUE:
                        option.mValue = parseString(tpos);
                        break;

                    case ATTR_ONPICK:
                        option.mOnPick = parseURL(tpos);
                        break;
                }
            } while(token != TOK_END);

            if(!hasContent(mCurrentToken))
                throw FatalParsingError(FatalParsingError::kContentExpected);

            mCurrOptionP->incrementOptionCount();

            WMLOption * optionP = new WMLOption(mGlyphStack.getLastElement(), option);

            mGlyphStack.add(optionP);
            mStateStack.add(mCurrentTextState);
        }
    }

    void
    WBXMLParser::tag_refresh(ULONG tpos)
    {
        // page 48
        WMLRefresh refresh;

        if(hasAttribute(mCurrentToken))
        {
            Byte token;
            do {
                token = parseByte(tpos);
                switch(token)
                {
                    case ATTR_REFRESH:
                        refresh.mRefresh = parseString(tpos);
                        break;
                }
            } while(token != TOK_END);

            if(!hasContent(mCurrentToken))
                throw FatalParsingError(FatalParsingError::kContentExpected);

            mCurrRefreshP->incrementRefreshCount();

            WMLRefresh * refreshP = new WMLRefresh(mGlyphStack.getLastElement(), refresh);

            mGlyphStack.add(refreshP);
            mStateStack.add(mCurrentTextState);
        }
    }
}

void
WBXMLParser::tag_refresh(ULONG tpos)
{
    // page 48
    WMLRefresh refresh;

    if(hasAttribute(mCurrentToken))
    {
        Byte token;
        do {
            token = parseByte(tpos);
            switch(token)
            {
                case ATTR_REFRESH:
                    refresh.mRefresh = parseString(tpos);
                    break;
            }
        } while(token != TOK_END);

        if(!hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kContentExpected);

        mCurrRefreshP->incrementRefreshCount();

        WMLRefresh * refreshP = new WMLRefresh(mGlyphStack.getLastElement(), refresh);

        mGlyphStack.add(refreshP);
        mStateStack.add(mCurrentTextState);
    }
}

```

```

(
    if (hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    WMLRefresh * refresh = new WMLRefresh(mGlyphStack.getLastElement());
    if (mCurrentTextState.action->setAction(refresh));
    if (!hasContent(mCurrentToken))
        return;
    mGlyphStack.add(refresh);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_select(ULONG &pos)
{
    // page 44
    WMLSelect select;
    if (hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    Byte token;
    do {
        token = parseByte(pos);
        switch(token)
        {
            case ATTR_TITLE:
                select.mTitle = parseString(pos);
                break;
            case ATTR_NAME:
                select.mName = parseString(pos);
                break;
            case ATTR_VALUE:
                select.mValue = parseString(pos);
                break;
            case ATTR_INAME:
                select.mName = parseString(pos);
                break;
            case ATTR_IValue:
                select.mValue = parseString(pos);
                break;
            case ATTR_TABINDEX:
                select.mTabIndex = (int) StrToInt(parseString(pos));
                break;
            case ATTR_MULTIPLE_TRUE:
                select.mType = WMLSelect::kCheckboxes;
                break;
            case ATTR_MULTIPLE_FALSE:
                select.mType = WMLSelect::kRadioButtons;
                break;
        }
        while(token != TOK_END);
    }
    if (!hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kContentExpected);
    // !!! TEMP CHANGE !!!
    select.mType = WMLSelect::kPopUpList;
    mCurrSelectP = new WMLSelect(mGlyphStack.getLastElement(), select);
}

mGlyphStack.add((!ParentGlyph *) mCurrSelectP);
mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_small(ULONG &pos)
{
    mCurrentTextState.bits.small = true;
    if (hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    mGlyphStack.add(mGlyphStack.getLastElement());
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_strong(ULONG &pos)
{
    mCurrentTextState.bits.strong = true;
    if (hasAttribute(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);
    mGlyphStack.add(mGlyphStack.getLastElement());
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_template(ULONG &pos)
{
    if (mRoot == NULL)
        throw FatalParsingError(FatalParsingError::kRootUndefined);
    WMLIntrinsicEvents events;
    CharPtr string;
    // !!! this leaks memory if we reach end of buffer in the middle
    // of this. So make sure WMLHyperlink isn't too big.
    Byte token;
    do {
        token = parseByte(pos);
        switch(token)
        {
            case ATTR_ONENTERFORWARD:
                string = parseString(pos);
                events.mOnEnterForward = new WMLHyperlink(string);
                break;
            case ATTR_ONENTERBACKWARD:
                string = parseString(pos);
                events.mOnEnterBackward = new WMLHyperlink(string);
                break;
            case ATTR_ONTIMER:
                string = parseString(pos);
                events.mOnTimer = new WMLHyperlink(string);
                break;
        }
        while(token != TOK_END);
    }
    WMLTemplate * templateP = new WMLTemplate(mGlyphStack.getLastElement());
    templateP->mEvents = events;
    mRoot->mTemplate = templateP;
}

```

```

    mGlyphStack.add(templateP);
    mStateStack.add(mCurrentTextState);
}

void
WBXMLParser::tag_timer(ULong &pos)
{
    ULong value = 0;

    if (hasAttribute(mCurrentToken))
    {
        Byte token;

        do {
            token = parseByte(pos);
            switch(token)
            {
                case ATTR_VALUE:
                    value = (ULong) StrToInt(parseString(pos));
                    break;
            }
        } while(token != TOK_END);

        if (hasContent(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kUnexpectedContent);

        BGLGlyph * foundGlyph;
        if (getEnclosingContext(foundGlyph) == TAG_CARD)
        {
            WMLTimer * timer = new WMLTimer(dynamic_cast<WMLCard *>(foundGlyph),
                value);
        }
    }

    void
    WBXMLParser::tag_u(ULong & /*pos*/)
    {
        mCurrentTextState.underline = solidUnderline;

        if (hasAttribute(mCurrentToken))
            throw FatalParsingError(FatalParsingError::kUnexpectedAttribute);

        mGlyphStack.add(mGlyphStack.getLastElement());
        mStateStack.add(mCurrentTextState);
    }

    void
    WBXMLParser::tag_setvar(ULong &pos)
    {
        // page 20

        WMLSetvar setvar;

        if (hasAttribute(mCurrentToken))
        {
            Byte token;

            do {
                token = parseByte(pos);
                switch(token)
                {
                    case ATTR_NAME:
                        setvar.mName = parseString(pos);
                        break;
                    case ATTR_VALUE:
                        setvar.mValue = parseString(pos);
                }
            }
        }
    }

```

```

        break;
    }
    while(token != TOK_END);

    if (hasContent(mCurrentToken))
        throw FatalParsingError(FatalParsingError::kUnexpectedContent);

    WMLSetvar * setvarp = new WMLSetvar(mGlyphStack.getLastElement(), setvar);
}

void
WBXMLParser::tag_wml(ULong & /*pos*/)
{
    mRoot = new WMLRoot();

    mGlyphStack.add(mRoot);
    mStateStack.add(mCurrentTextState);
}

// WBXMLParser.h
#pragma once

#include "XMLParser.h"
#include "BTypes.h"
#include "BGLGlyph.h"
#include "WMLState.h"
#include "WBXMLConstants.h"
#include "BLWMLInput.h"

class UnexpectedEOF
{
public:
    UnexpectedEOF(ULong pos = 0, VoidPtr datap = NULL) { mPos = pos; mData = datap; }

    ULong mPos;
    VoidPtr mData;
};

class FatalParsingError
{
public:
    enum FatalCode {
        kUnknownError = 0,
        kFileTypeError,
        kRootUndefined,
        kUnexpectedContent,
        kUnexpectedAttribute,
        kUnexpectedEndOfStack,
        kContentExpected };

    FatalParsingError(FatalCode code) { mCode = code; }

    FatalCode mCode;
};

class TopItem : public BGLGlyph
{
public:
    class WBXMLParser : public XMLParser
    {
    public:
        WBXMLParser();
        ~WBXMLParser();

        virtual ULong readData(CharPtr buffer, ULong length);

        WMLRoot * getRoot() { return mRoot; }
    }
}

```

```

// !!! These don't really need to be static. This is just the
// easiest way for flow() to get to these.
static BGlyph * sGetCurrentNode() { return sCurrentGlyph; }
static int sGetNumOpenTables() { return sOpenTables; }

BVector<WMLImage *, BGlyph::kChildrenInc> & getImages() { return mImages; }

protected:
void setCurrGlyph();

WMLTagToken getEnclosingContext(BGLGlyph * &foundGlyph);

ULONG parseMultiByteInt(ULONG &pos);
Byte parseByte( ULONG &pos);
CharPtr parseString( ULONG &pos, ULONG * lengthP = NULL, bool parseType = true);
CharPtr parseURL( ULONG &pos);
ULONG stringLength( ULONG &pos);
void parseHeader( ULONG &pos);

bool hasAttribute( Byte token) { return (token & 0x80) != 0; }
bool hasContent( Byte token) { return (token & 0x40) != 0; }

bool bool run( ULONG &pos);
void unknown_tag( ULONG &pos);

void tag_a( ULONG &pos);
void tag_td( ULONG &pos);
void tag_tr( ULONG &pos);
void tag_table( ULONG &pos);

void tag_p( ULONG &pos);
void tag_postfield( ULONG &pos);
void tag_anchor( ULONG &pos);
void tag_access( ULONG &pos);
void tag_big( ULONG &pos);
void tag_br( ULONG &pos);
void tag_card( ULONG &pos);
void tag_do( ULONG &pos);
void tag_em( ULONG &pos);
void tag_fieldset( ULONG &pos);
void tag_go( ULONG &pos);
void tag_head( ULONG &pos);
void tag_img( ULONG &pos);
void tag_input( ULONG &pos);

void tag_meta( ULONG &pos);
void tag_noop( ULONG &pos);
void tag_prev( ULONG &pos);
void tag_onevent( ULONG &pos);
void tag_optgroup( ULONG &pos);
void tag_option( ULONG &pos);
void tag_refresh( ULONG &pos);
void tag_select( ULONG &pos);
void tag_small( ULONG &pos);
void tag_strong( ULONG &pos);

void tag_template( ULONG &pos);
void tag_timer( ULONG &pos);
void tag_u( ULONG &pos);
void tag_setvar( ULONG &pos);
void tag_wml( ULONG &pos);

ULONG mPublicID;
Byte mid_index;
CharPtr mStringTable;
ULONG mCharSet;
ULONG mStringTableLen;
CharPtr mBuffer;
ULONG mPos;
ULONG mLength;

Byte mCurrentToken;
TextState mCurrentTextState;
BVector<BGLGlyph *, BGlyph::kChildrenInc> mGlyphStack;

```

```

BVector<TextState, BGlyph::kChildrenInc> mStateStack;
BVector<WMLImage *, BGlyph::kChildrenInc> mImages;

bool mHeaderDone;
CharPtr mCurrTextBuffer;
ULONG mCurrTextBufferLen;
WMLSelect * mCurrSelectP;

static BGlyph * sCurrentGlyph;
static int sOpenTables;

typedef void (WBXMLParser::TagHandlerType)(ULONG &pos); <?xml version="1.0"?>
<?code>error export version="1.0" id version="4.0"?>
<!DOCTYPE PROJECT [
<ELEMENT PROJECT (TARGETLIST, TARGETORDER, GROUPLIST, DESIGNLIST?)>
<ELEMENT TARGETLIST (TARGET+)>
<ELEMENT TARGET (NAME, SETTINGLIST, FILELIST?, LINKORDER?, SEGMENTLIST?, OVERLAYGROUPLIST?)>
SUBTARGETLIST?, SUBPROJECTLIST?)>
<ELEMENT NAME (#PCDATA)>
<ELEMENT USERSOURCETYPE (#PCDATA)>
<ELEMENT PATH (#PCDATA)>
<ELEMENT FILELIST (FILE+)>
<ELEMENT FILE (PATHTYPE, PATHROOT?, ACCESSPATH?, PATH, PATHFORMAT?, ROOTFILEREFF?,
FILEKIND?, FILEFLAGS?)>
<ELEMENT PATHTYPE (#PCDATA)>
<ELEMENT PATHROOT (#PCDATA)>
<ELEMENT ACCESSPATH (#PCDATA)>
<ELEMENT PATHFORMAT (#PCDATA)>
<ELEMENT ROOTFILEREFF (PATHTYPE, PATHROOT?, ACCESSPATH?, PATH, PATHFORMAT?)>
<ELEMENT FILEKIND (#PCDATA)>
<ELEMENT FILEFLAGS (#PCDATA)>
<ELEMENT FILEREFF (TARGETNAME?, PATHTYPE, PATHROOT?, ACCESSPATH?, PATH, PATHFORMAT?)>
<ELEMENT TARGETNAME (#PCDATA)>
<ELEMENT SETTINGLIST ((SETTING|PANELDATA)+)>
<ELEMENT SETTING (NAME?, (VALUE|(SETTING+)))>
<ELEMENT PANELDATA (NAME, VALUE)>
<ELEMENT VALUE (#PCDATA)>
<ELEMENT LINKORDER (FILEREFF+)>
<ELEMENT SEGMENTLIST (SEGMENT+)>
<ELEMENT SEGMENT (NAME, ATTRIBUTES?, FILEREFF+)>
<ELEMENT ATTRIBUTES (#PCDATA)>
<ELEMENT OVERLAYGROUPLIST (OVERLAYGROUP+)>
<ELEMENT OVERLAYGROUP (NAME, BASEADDRESS, OVERLAY+)>
<ELEMENT BASEADDRESS (#PCDATA)>
<ELEMENT OVERLAY (NAME, FILEREFF+)>
<ELEMENT SUBTARGETLIST (SUBTARGET+)>
<ELEMENT SUBTARGET (TARGETNAME, ATTRIBUTES?)>
<ELEMENT SUBPROJECTLIST (SUBPROJECT+)>
<ELEMENT SUBPROJECT (FILEREFF, SUBPROJECTTARGETLIST)>
<ELEMENT SUBPROJECTTARGETLIST (SUBPROJECTTARGET+)>
<ELEMENT SUBPROJECTTARGET (TARGETNAME, ATTRIBUTES?)>
<ELEMENT TARGETORDER (ORDEREDTARGET|ORDEREDDESIGN)+>
<ELEMENT ORDEREDTARGET (NAME)>
<ELEMENT ORDEREDDESIGN (NAME, ORDEREDTARGET+)>
<ELEMENT GROUPLIST (GROUP|FILEREFF+)>
<ELEMENT GROUP (NAME, GROUP|FILEREFF+)>
<ELEMENT DESIGNLIST (DESIGN+)>
<ELEMENT DESIGN (NAME, DESIGNDATA)>
<ELEMENT DESIGNDATA (#PCDATA)>
]>
</PROJECT>
<TARGETLIST>
<TARGET>
<NAME>Blazer</NAME>
<SETTINGLIST>
<!-- Settings for "Source Trees" panel -->
<SETTING<NAME>UserSourceTrees</NAME><VALUE></VALUE></SETTING>
<!-- Settings for "Custom Keywords" panel -->

```

```
<SETTING><NAME>PathRoot</NAME><VALUE>CodeWarrior</VALUE></SETTING>
</SETTING>
<SETTING><NAME>Recursive</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>HostFlags</NAME><VALUE>All</VALUE></SETTING>
</SETTING>
<SETTING><NAME>SearchPath</NAME>
<SETTING><NAME>Path</NAME><VALUE>Palm VII
Support\Incs\System</VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>Windows</VALUE></SETTING>
</SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>CodeWarrior</VALUE></SETTING>
</SETTING>
<SETTING><NAME>Recursive</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>HostFlags</NAME><VALUE>All</VALUE></SETTING>
</SETTING>
<SETTING><NAME>SearchPath</NAME>
<SETTING><NAME>Path</NAME><VALUE>Palm OS 3.1
Support</VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>Windows</VALUE></SETTING>
</SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>CodeWarrior</VALUE></SETTING>
</SETTING>
<SETTING><NAME>Recursive</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>HostFlags</NAME><VALUE>All</VALUE></SETTING>
</SETTING>
<!-- Settings for "Target Settings" panel -->
<SETTING><NAME>Linker</NAME><VALUE>MacOS 68K Linker</VALUE></SETTING>
<SETTING><NAME>PreLinker</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>PostLinker</NAME><VALUE>PalmRez Post Linker</VALUE></SETTING>
<SETTING><NAME>TargetName</NAME><VALUE>Blazer</VALUE></SETTING>
<SETTING><NAME>OutputDirectory</NAME>
<SETTING><NAME>Path</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>MacOS</VALUE></SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>Project</VALUE></SETTING>
</SETTING>
<SETTING><NAME>SaveEntriesUsingRelativePaths</NAME><VALUE>false</VALUE></SETTING>
<!-- Settings for "File Mappings" panel -->
<SETTING><NAME>FileMappings</NAME>
<SETTING><NAME>FileType</NAME><VALUE>APPL</VALUE></SETTING>
<SETTING><NAME>FileExtension</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>Compiler</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>Precompile</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>Launchable</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>ResourceFile</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>IgnoredByMake</NAME><VALUE>false</VALUE></SETTING>
</SETTING>
<SETTING><NAME>FileType</NAME><VALUE>Appl</VALUE></SETTING>
<SETTING><NAME>FileExtension</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>Compiler</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>Precompile</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>Launchable</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>ResourceFile</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>IgnoredByMake</NAME><VALUE>false</VALUE></SETTING>
</SETTING>
<SETTING><NAME>FileType</NAME><VALUE>MMLB</VALUE></SETTING>
<SETTING><NAME>FileExtension</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>Compiler</NAME><VALUE>Lib Import
68K</VALUE></SETTING>
<SETTING><NAME>Precompile</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>Launchable</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>ResourceFile</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>IgnoredByMake</NAME><VALUE>false</VALUE></SETTING>
</SETTING>
<SETTING><NAME>FileType</NAME><VALUE>MPLF</VALUE></SETTING>
<SETTING><NAME>FileExtension</NAME><VALUE></VALUE></SETTING>
</SETTING>
<!-- Settings for "Access Paths" panel -->
<SETTING><NAME>AlwaysSearchUserPaths</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>InterpretDOSAndUnixPaths</NAME><VALUE>false</VALUE></SETTING>
<SETTING><NAME>UserSearchPaths</NAME>
<SETTING><NAME>SearchPath</NAME>
<SETTING><NAME>Path</NAME><VALUE></VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE></VALUE></SETTING>
</SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>MacOS</VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>Windows</VALUE></SETTING>
</SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>CodeWarrior</VALUE></SETTING>
</SETTING>
<SETTING><NAME>Recursive</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>HostFlags</NAME><VALUE>All</VALUE></SETTING>
</SETTING>
<SETTING><NAME>SearchPath</NAME>
<SETTING><NAME>Path</NAME><VALUE>MSL\MSL_C\MSL_Common\Include</VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>Windows</VALUE></SETTING>
</SETTING>
<SETTING><NAME>PathRoot</NAME><VALUE>CodeWarrior</VALUE></SETTING>
</SETTING>
<SETTING><NAME>Recursive</NAME><VALUE>true</VALUE></SETTING>
<SETTING><NAME>HostFlags</NAME><VALUE>All</VALUE></SETTING>
</SETTING>
<SETTING><NAME>SearchPath</NAME>
<SETTING><NAME>Path</NAME><VALUE>MSL\MSL_C++\MSL_Common\Include</VALUE></SETTING>
<SETTING><NAME>PathFormat</NAME><VALUE>Windows</VALUE></SETTING>
</SETTING>
```





[illegible]

Page: 197

[illegible]









```
<PATH>Name</PATH>
<PATH>BLMMLInput.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
<FILEKIND>Text</FILEKIND>
<FILEFLAGS>Debug</FILEFLAGS>
</FILE>
<FILE>
<PATH>Name</PATH>
<PATH>WMLVariables.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
<FILEKIND>Text</FILEKIND>
<FILEFLAGS></FILEFLAGS>
</FILE>
<FILE>
<PATH>Name</PATH>
<PATH>WMLVariables.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
<FILEKIND>Text</FILEKIND>
<FILEFLAGS>Debug</FILEFLAGS>
</FILE>
</FILELIST>
<SEGMENTLIST>
<SEGMENT>
<NAME>Main Segment</NAME>
<ATTRIBUTES>Protected, Purgeable</ATTRIBUTES>
</SEGMENT>
<PATH>Name</PATH>
<PATH>PrecompiledHeaders.pch</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLMMLSession.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BrowserRcl.rsrc</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>NetSocket.c</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>Main.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BrowserApp.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BrowserApp.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLMMLView.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLMMLView.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLTypes.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
```

```
</FILEREF>
<PATH>Name</PATH>
<PATH>HTTPRequest.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>HTTPRequest.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLNetworking.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLNetworking.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>MSL Runtime Palm OS (21) .lib</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>WXMLConstants.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLCacheDB.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLDatabase.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLDatabase.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>WMLState.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>WMLState.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BrowserRcl_rsc.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLWMLInput.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>BLWMLInput.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</FILEREF>
<PATH>Name</PATH>
<PATH>WMLVariables.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
```

```
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>WMVariables.cpp</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
</SEGMENT>
<SEGMENT>
  <NAME>Bli</NAME>
  <ATTRIBUTES>Protected, Purgeable</ATTRIBUTES>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BMemoryManager.cpp</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLMMLTable.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLMMLTable.cpp</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLVector.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLCPrefs.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>XMLParser.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLCacheDB.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLApplication.cpp</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLIterator.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLForm.h</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
<FILEREF>
  <PATHTYPE>Name</PATHTYPE>
  <PATH>BLForm.cpp</PATH>
  <PATHFORMAT>Windows</PATHFORMAT>
</FILEREF>
```

```
<PATHTYPE>Name</PATHTYPE>
<PATH>BLeventhHandler.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BGLlyph.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BGLlyph.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BStack.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLView.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLView.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<SEGMENT>
<ATTRIBUTES>Protected, Purgeable</ATTRIBUTES>
<PATHTYPE>Name</PATHTYPE>
<PATH>WBXMLParser.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>WBXMLParser.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLURLHandler.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLMWLForm.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLMWLForm.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLMWLSession.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLUtils.cpp</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLUtils.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
<PATH>BLUtils.h</PATH>
<PATHFORMAT>Windows</PATHFORMAT>
</FILEREFF>
<PATHTYPE>Name</PATHTYPE>
```









```

</GROUPLIST>
</PROJECT>
// WMLState.cpp
#include "BLWMLActions.h"
LineMetrics::LineMetrics()
{
    MemSet(this, sizeof(*this), 0);
}
int width;
int baselineOffset;
int overhang;
int leadingSpace;
};

```

```

class TextState
{
public:
    TextState();
    static TextState getBasicState() { TextState state; return state; }
    static void enableBasicState() { TextState state; state.enable(); }
    void enable();
    void enable(const TextState &previous);
    void setWordWrap(bool isOn)
        ( if(bits.wordWrap && !isOn) bits.wordWrap = false; )
};

```

```

UnderlineModeType underline;
struct
{
    bool wordWrap : 1;
    bool italic : 1;
    bool strong : 1;
    bool emphasis : 1;
    bool big : 1;
    bool small : 1;
    bool preformatted : 1;
    } bits;
};

```

```
WMLAction * action;
```

```
};
```

```

class BLHistoryItem
{
public:
    BLHistoryItem(CharPtr url, CharPtr cardID);
    ~BLHistoryItem();
    CharPtr getURL() { return mURL; }
    CharPtr getCardID() { return mCardID; }
protected:
    CharPtr mURL;
    CharPtr mCardID;
};

```

```

enum NavigationAction { kNone = 0, kGoTo, kGoBack, kGoForward };
// WMLVariables.cpp
#include "WMLVariables.h"
#include "BLUtils.h"
const int kInitNumBuckets = 17;
const int kNameBufLen = 256;
WMLVariables::WMLVariables()
{
    mNumBuckets = kInitNumBuckets;
    mBuckets = new VarEntry*[mNumBuckets];
    for(int i=0; i<mNumBuckets; i++)
    {
        mBuckets[i] = NULL;
    }
}

```

```

void
TextState::enable(const TextState &previous)
{
    // !!! future optimization: only turn on distinct states
    enable();
}
BLHistoryItem::BLHistoryItem(CharPtr url, CharPtr cardID)
{
    mURL = url;
    mCardID = cardID;
}
BLHistoryItem::~BLHistoryItem()
{
    delete[] mURL;
    delete[] mCardID;
}
// WMLState.h
#pragma once
#include "BLTypes.h"
class WMLAction;
class LineMetrics
{
public:
    LineMetrics();
    Point getExtent() { return Point(width, height()); }
    int height() { return baselineOffset + overhang; }
};

```

```

WMLVariables::~WMLVariables()
{
    delete [] mBuckets;
}

VarEntry *
WMLVariables::findEntry(CharPtr key)
{
    VarEntry * entry = mBuckets[hashFunction(key)];

    while(entry != NULL)
    {
        if(StrCompare(key, entry->name) == 0)
            return entry;

        entry = entry->next;
    }

    return NULL;
}

CharPtr
WMLVariables::lookup(CharPtr key)
{
    if(key == NULL)
        return NULL;

    VarEntry * entry = findEntry(key);

    if(entry)
        return entry->value;
    else
        return NULL;
}

void
WMLVariables::add(CharPtr key, CharPtr value)
{
    if((key == NULL) || (value == NULL))
        return;

    VarEntry * entry = findEntry(key);

    if(entry != NULL)
    {
        delete [] entry->value;

        entry->value = BUUtils::cloneString(value);
        return;
    }

    UInt bucketNum = hashFunction(key);

    entry = new VarEntry( BUUtils::cloneString(key),
                          BUUtils::cloneString(value));

    if(entry == NULL)
        return;

    entry->next = mBuckets[bucketNum];
    mBuckets[bucketNum] = entry;
}

UInt
WMLVariables::hashFunction(CharPtr key)
{
    UInt hash;
    UInt len = StrLen(key);
    UInt i;

```

```

        for(hashLen = 1; i < len; i++)
            hash = (hash << 4) ^ (hash >> 28) ^ key[i];

        return (UInt)(hash % (ULong)mNumBuckets);
    }

    enum VarState { kLooking, kDollarSign, kOpenParen, kVariable };

    bool
    WMLVariables::insertVariables( CharPtr destination,
                                   const CharPtr source,
                                   int destBufferLength)
    {
        UInt pos = 0;
        UInt destPos = 0;
        UInt varPos = 0;

        Char varName[kNameBufLen];

        VarState state = kLooking;

        while(source[pos])
        {
            if((destPos+1) >= destBufferLength)
                return false;

            switch(state)
            {
                case kLooking:
                    if(source[pos] == '$')
                    {
                        state = kDollarSign;
                        break;
                    }

                    destination[destPos++] = source[pos];
                    break;

                case kDollarSign:
                    if(source[pos] == '$')
                    {
                        state = kLooking;
                        destination[destPos++] = source[pos];
                        break;
                    }

                    else if(source[pos] == '(')
                    {
                        state = kOpenParen;
                        varPos = 0;
                        break;
                    }

                    else
                    {
                        state = kVariable;
                        varPos = 0;
                        varName[varPos++] = source[pos];
                        break;
                    }

                case kOpenParen:
                    case kVariable:
                    if( (kOpenParen && (source[pos] == '(')) ||
                        (kVariable && BUUtils::isWhitespace(source[pos])) )
                    {
                        varName[varPos] = '\0';
                        CharPtr value = lookup(varName);

                        if(value)
                        {
                            if(BUUtils::copyStringIfRoom(destination + destPos, value,
                                                            destBufferLength - (int)destPos))
                                return false;

                            destPos += StrLen(value);
                        }
                    }

```

```

    }
    state = kLooking;
    break;
}
else if(source[pos] == ':')
{
    varName[varPos++] = '\0';
    break;
}
else
{
    varName[varPos++] = source[pos];
    break;
}
}
pos++;
if(state == kInVariable)
{
    pos--;
    goto SUB_VAR;
}
destination[destPos] = '\0';
return true;
}

// WMLVariables.h
class VarEntry
{
public:
    VarEntry(CharPtr nm, CharPtr val) { name = nm; value = val; }

    CharPtr name;
    CharPtr value;

    VarEntry * next;
};

class WMLVariables
{
public:
    WMLVariables();
    ~WMLVariables();

    CharPtr lookup(CharPtr key);
    void add(CharPtr key, CharPtr value);

    bool insertVariables(CharPtr sdestination, const CharPtr source, int
destBufferLength);

protected:
    VarEntry * findEntry(CharPtr key);

    UInt hashFunction(CharPtr key);

    int mNumBuckets;
    VarEntry ** mBuckets;
};

// WBXMLParser.h
#pragma once
#include "BLVector.h"

```

```

#include "BLTypes.h"
#include "BLMemoryManager.h"
class WMLRoot;

class XMLParser : public DataReader
{
public:
    virtual ULONG readData(CharPtr buffer, ULONG length) = 0;
    WMLRoot * getRoot() { return mRoot; }

protected:
    WMLRoot * mRoot;
};

```

**APPENDIX D**

**Functional Requirements Document**

**APPENDIX D**

**Functional Requirements Document**

**APPENDIX D**

**Functional Requirements Document**

# **BLUESKY**

## **MOBILE APPLICATION SERVER**

### **Functional Requirements Document**

**Version 0.1, 1/30/2001**

#### Revision notes

Version	Date	Author	Comments
0.1	1/30/01	Katie	Written based on my interpretation of Ben's brain dump. Still needs attachments for ZML DTD and list of supported elements of HTML and WML



1	OVERVIEW .....	3
2	PURPOSE .....	3
3	CONTENT TRANSFORMATION MODULE (CTM).....	3
3.1	TRANSCODING: HTML, HDML, WML, & cHTML TO “ZML” .....	3
3.1.1	“ZML” .....	3
3.1.2	HTML or cHTML to ZML.....	4
3.1.3	HDML & WML to ZML.....	5
3.2	IMAGES .....	5
3.2.1	Scaling image height and width .....	5
3.2.2	Reducing image bit depth; convert to Palm bitmap .....	5
4	PROXY MODULE.....	6
4.1	USER ACCOUNTS .....	6
4.2	COOKIES .....	6
4.3	SERVER REDIRECTS .....	6
4.4	SECURITY .....	6

[illegible]



# 1 Overview

Handspring's BlueSky mobile application server includes two main components, an HTTP proxy server module (proxy) and a content transformation module (CTM). The proxy is a generic HTTP proxy that accepts requests for Web pages from the Blazer client and passes those requests on to remote Web servers. Once Web content is returned to the proxy from those remote Web servers, it is dynamically run through a series of transformations on the CTM, then transformed content is streamed back to the Blazer client. These two modules are further described below.

## 2 Purpose

The BlueSky mobile application server provides users of mobile devices with the ability to view a wide variety of Web content. Currently, the majority of Internet content is designed to be viewed on desktop computers. Most mobile devices differ dramatically from desktop computers with regard to screen size, processor power, and color capabilities. The BlueSky server optimizes Web content to improve the wireless Web experience for users of mobile devices. First, the proxy server acts as a connected agent on behalf of mobile users, storing important personal and account information and managing much of the "heavy lifting" that would slow down Web access in the absence of the proxy. To augment the value of the proxy, the CTM optimizes pages designed for desktop use to viewing on a smaller screen device.

## 3 Content Transformation Module (CTM)

The purpose of the CTM is to optimize Web content for viewing on a small screen device. The CTM adapts content to the screen size and display capabilities of the device. Currently, this adaptation is geared toward screens that measure 160x160 pixels and to devices that run on the Palm operating system *[is there anything specific to Palm OS in the CTM?]*

The CTM actually consists of a series of discrete transformations that convert Web documents and images to formats that are understandable to a Palm OS device running the Blazer browser. Those transformations include transcoding HTML, HDML, WML and cHTML documents to "ZML" and adapting images to meet the device capabilities. The following is a description of those processes.

### 3.1 Transcoding: HTML, HDML, WML, & cHTML to "ZML"

The CTM transcodes documents from several markup languages to "ZML," a language unique to Blazer that is a superset of WML.

#### 3.1.1 "ZML"

Blazer natively displays documents marked up in "ZML," a language unique to Blazer that is a superset of WML. ZML supports all elements in the WML 1.1 specification, including Phone.com (now OpenWave) language extensions. In addition, ZML also supports additional extensions that optimize support for HTML pages. Following is a list of the extensions supported by ZML *[Need to fill in list]:*

	Elements	Attributes	Description
phone.com	<link>		
phone.com	<spawn>		
phone.com	<exit>		
phone.com	<catch>		
phone.com	<throw>		
	<hr>		inserts a horizontal line
	<base>		

	<form>		
--	--------	--	--

### 3.1.2 HTML or cHTML to ZML

The majority of pages accessed by most Blazer users are HTML pages. HTML pages go through a series of steps as they are transcoded to ZML pages. These steps are outlined below:

#### 3.1.2.1 Step One: Convert HTML to valid, well-formed XML document

The first step in the transcoding of HTML to ZML is to convert the HTML into a valid, well-formed XML document. To do this, the CTM uses a generic algorithm to "normalize" the markup language to ensure that the HTML document conforms to the HTML Document Type Description (DTD). During this process, the CTM "fixes" HTML that does not conform to the DTD. Well-formed XML documents must include the following (from URL: <http://www.ucc.ie/xml/>):

- All tags must be balanced: that is, all elements which may contain character data must have both start- and end-tags present (omission is not allowed except for empty elements, see below);
- All attribute values must be in quotes (the single-quote character [the apostrophe] may be used if the value contains a double-quote character, and vice versa): if you need both, use &apos; or &quot; (do *not* under any circumstances use the typographic (curly) 'inverted commas' for quoting attribute values);
- Any EMPTY element tags (e.g. those with no end-tag like HTML's <IMG>, <HR>, and <BR> and others) must either end with '/' or you have to make them appear non-EMPTY by adding a real end-tag; Example: <BR> would become either <BR/> or <BR></BR>.
- There must not be any isolated markup-start characters (< or &) in your text data (ie they must be given as &lt; and &amp;), and the sequence >> must be given as >>&gt; unless you really are using it as the end of a CDATA marked section;
- Elements must nest inside each other properly (no overlapping markup, same rule as for all SGML, including HTML);
- Well-formed documents with no DTD may use attributes on any element, but the attributes are assumed to be all of type CDATA. You cannot use ID/IDREF attributes in DTDless documents.

In addition to using the generic algorithm to normalize the document, the CTM applies additional rules to manage special cases. These additional rules are outlined below [*need details on additional rules*]:

#### 3.1.2.2 Step Two: Strip out unsupported content

Once the HTML has been converted to a valid, well-formed XML document, the CTM strips out content that cannot be supported by the Blazer browser. Supported content includes a subset of the HTML 3.2 DTD and the WML 1.1 DTD including the Phone.com/OpenWave extensions (see attached). Examples of elements that are not supported include: scripts (JavaScript, VBscript), Java applets, plug-ins (e.g. Flash, Shockwave, RealAudio) [*I'm sure there are more here...*]

The list of elements and attributes that are supported is subject to change regularly.

#### 3.1.2.3 Step Three: Reformat content for layout on screen

The CTM applies a number of rules to the content to optimize for viewing on a small-screen device. For example, the CTM strips out extra white space by removing extra line break tags (<BR>), changing line breaks to paragraph tags, stripping out empty paragraph tags, stripping out line break tags at the beginning or end of table cells, etc. In addition, the CTM unrolls large page-level tables and displays elements from within the table stacked (?) on a page. The CTM preserves certain nested tables that fit within the device screen constraints. [*Not including further details about this...trade secret :)*]

### 3.1.2.4 Step Four: Convert to ZML

To convert the valid, well-formed XML document to ZML, the CTM changes HTML tags to their equivalent ZML tags (WML + phone.com language extensions + additional extensions). For example, the header tags (<H1, H2, etc.) are not supported in ZML. In this case, the CTM changes the header tags to equivalent WML tags that display text big and bold (<h1>text</h1> to <big><b>text</b></big>).

### 3.1.2.5 Step Five: Convert ZML to ZML-C

The ZML created by the CTM is then converted to binary (tokenized) ZML-C according to the specifications for WBXML before it is sent to the device. This compact binary representation of ZML reduces the transmission size of the document with no loss of functionality or semantic information.

### 3.1.2.6 Special treatment of sites identified as optimized for handheld devices

Web pages that include meta tags designating them as optimized for handheld devices are treated differently. These pages are typically marked up in HTML and are geared toward users of Palm and PocketPC devices. They include the following tags in their headers: <meta name="HandheldFriendly" content="true"> OR <meta name=PalmComputingPlatform content=true>.

When parsing (?) handheld-friendly sites, the CTM preserves all tables unaltered (i.e. does not unroll tables). It also converts certain types of URLs (found in Palm PQAs) to URLs that point to appropriate files on the content Web server. Lastly, the CTM converts certain elements used in PQA pages (DeviceID and %zipcode) to WML-style variables.

### 3.1.3 HDML & WML to ZML

The CTM also converts HDML (pre-cursor to WML; still widely used for site compatibility with all phone.com UP browsers) to WML and WML to ZML. There are very few changes that happen during this conversion as ZML is essentially WML with certain extensions. [*What's involved in converting HDML to WML?*]

## 3.2 Images

In a parallel process, the CTM converts images into a format that is supported by the device. This process is currently geared toward Palm OS device screens and capabilities (e.g., screens with 160x160 pixels with 7 pixel scrollbar, particular image bit depths).

### 3.2.1 Scaling image height and width

Currently, the CTM reduces the size of any image that exceeds 153 pixels in height or width. The image is scaled proportionally until the longer of the two dimensions is exactly 153 pixels. We anticipate that future versions of the CTM will apply additional intelligence to image scaling. For example, many Web sites place several images flush next to one another horizontally on the screen to represent a single image (e.g. a navigation bar). The CTM should recognize these images as linked and scale them all by the same factor so that they maintain their position on the screen.

### 3.2.2 Reducing image bit depth; convert to Palm bitmap

The CTM also converts the image to the bit depth that is supported by the device and specified by the user preference. This information is communicated to the CTM when the browser makes its request. Images are also converted from several formats (GIF, JPG, WBMP, PNG, BMP) to Palm bitmaps.

## 4 Proxy module

The BlueSky proxy module is a generic HTTP proxy. It includes a few features that enhance the base functionality of an HTTP proxy, including establishing and verifying user account information, intercepting and storing cookies, handling server redirects, and enabling 128bit RSA security to secure remote Web sites. There is some possibility that we may opt to use a commercially available proxy for much of the base functionality and add support for user accounts, cookies, server redirects, and security to that product.

### 4.1 User accounts

The proxy requires a username and password to authenticate client devices. Users establish this username and password through an account setup on the BlueLark (later, Handspring) Web site. The username must be unique (in the future it will be email address) and the password must be an alphanumeric string greater than five characters. The username and password information is stored in an Oracle database and links the user to their stored cookies (see below). *[Anything else?]*

### 4.2 Cookies

The proxy module intercepts cookies that are sent from the remote Web server and stores them in an Oracle database. Typically, these cookies reside in a text file on the client device. There are several reasons that the proxy intercepts cookies. First, storing cookies on the server preserves valuable storage memory on the device. Second, if a user is forced to do a hard reset of their device, or if they use Blazer from another device or upgrade to a new device, this information continues to be available to them. Third, storing cookies on the proxy provides for faster access to sites that require cookies. The server is able to attach individual user cookies to requests that need them. *[What other reasons are there?]*

### 4.3 Server redirects

The proxy module also handles server redirects for the browser to improve the speed of access to the desired Web page. Rather than send data to the browser and require the browser to re-request the page, the server handles this without notifying the browser. The proxy has a limit on the total number of sequential redirects of xx.

### 4.4 Security

The proxy provides HTTPS support for secure connections, offering 128 bit elliptical curve cryptography (ECC) between the client and the proxy server and RSA encryption between the proxy and the target Web site. The encryption is maintained through the data carrier network. Data is unencrypted on the proxy and reencrypted before it is sent to the client device. Security is maintained through physical and electronic means (firewall, fortress???)

## ADAPTING TO NETWORK AND CLIENT VARIATIONS USING INFRASTRUCTURAL PROXIES: LESSONS AND PERSPECTIVES

[illegible]

**ADAPTING TO NETWORK AND CLIENT  
VARIATION USING INFRASTRUCTURAL PROXIES:  
LESSONS AND PERSPECTIVES**

0933404 : 064604  
005400

# Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives

Armando Fox      Steven D. Gribble      Yatin Chawathe  
Eric A. Brewer  
{fox,gribble,yatin,brewer}@cs.berkeley.edu

Today's Internet clients vary widely with respect to both hardware and software properties: screen size, color depth, effective bandwidth, processing power, and the ability to handle different data formats. The order-of-magnitude span of this variation is too large to hide at the network level, making application-level techniques necessary. We show that on-the-fly adaptation by transformational proxies is a widely applicable, cost-effective, and flexible technique for addressing all these types of variation. To support this claim, we describe our experience with datatype-specific distillation (lossy compression) in a variety of applications. We also argue that placing adaptation machinery in the network infrastructure, rather than inserting it into end servers, enables incremental deployment and amortization of operating costs. To this end, we describe a programming model for large-scale interactive Internet services and a scalable cluster-based framework that has been in production use at UC Berkeley since April 1997. We present a detailed examination of TranSend, a scalable transformational Web proxy deployed on our cluster framework, and give descriptions of several handheld-device applications that demonstrate the wide applicability of the proxy-adaptation philosophy.

## 1 Infrastructural On-the-Fly Adaptation Services

### 1.1 Heterogeneity: Thin Clients and Slow Networks

The current Internet infrastructure includes an extensive range and number of clients and servers. Clients vary along many axes, including screen size, color depth, effective bandwidth, processing power, and ability to handle specific data encodings, e.g., GIF, PostScript, or MPEG. As shown in tables 1 and 2, each type of variation often spans orders

of magnitude. High-volume devices such as smart phones [12] and smart two-way pagers will soon constitute an increasing fraction of Internet clients, making the variation even more pronounced.

These conditions make it difficult for servers to provide a level of service that is appropriate for every client. Application-level adaptation is required to provide a *meaningful* Internet experience across the range of client capabilities. Despite continuing improvements in client computing power and connectivity, we expect the high end to advance roughly in parallel with the low end, effectively maintaining a gap between the two and therefore the need for application-level adaptation.

Platform	SPEC92/ Memory	Screen Size	Bits/ pixel
High-end PC	200/64M	1280x1024	24
Midrange PC	160/32M	1024x768	16
Typ. Laptop	110/16M	800x600	8
Typical PDA	low/2M	320x200	2

Table 1: Physical variation among clients

Network	Bandwidth (bits/s)	Round-Trip Time
Local Ethernet	10-100 M	0.5 - 2.0 ms
ISDN	128 K	10-20 ms
Wireline Modem	14.4 - 56 K	350 ms
Cellular/CDPD	9.6 - 19.2 K	0.1 - 0.5 s

Table 2: Typical Network Variation

### 1.2 Approach: Infrastructural Proxy Services

We argue for a *proxy-based approach* to adaptation, in which proxy agents placed between clients and servers perform aggressive computation and storage on behalf of clients. The proxy ap-

proach stands in contrast to the *client-based approach*, which attempts to bring all clients up to a least-common-denominator level of functionality (e.g. text-only, HTML-subset compatibility for thin-client Web browsers), and the *server-based approach*, which attempts to insert adaptation machinery at each end server. We believe the proxy approach directly confers three advantages over the client and server approaches.

- **Leveraging the installed infrastructure through incremental deployment.** The enormous installed infrastructure, and its attendant base of existing content, is too valuable to waste; yet some clients cannot handle certain data types effectively. A compelling solution to the problem of client and network heterogeneity should allow interoperability with existing servers, thus enabling incremental deployment while evolving content formats and protocols are tuned and standardized for different target platforms. A proxy-based approach lends itself naturally to transparent incremental deployment, since an application-level proxy appears as a server to existing clients and as a client to existing servers.
- **Rapid prototyping during turbulent standardization cycles.** Software development on "Internet time" does not allow for long deployment cycles. Proxy-based adaptation provides a smooth path for rapid prototyping of new services, formats, and protocols, which can be deployed to servers (or clients) later if the prototypes succeed.
- **Economy of scale.** Basic queueing theory shows that a large central (virtual) server is more efficient in both cost and utilization (though less predictable in per-request performance) than a collection of smaller servers; standalone desktop systems represent the degenerate case of one "server" per user. This supports the argument for Network Computers [28] and suggests that collocating proxy services with infrastructural elements such as Internet points-of-presence (POPs) is one way to achieve effective economies of scale.

Large-scale network services remain difficult to deploy because of three fundamental challenges: scalability, availability and cost effectiveness. By scalability, we mean that when the load offered to the service increases, an incremental and linear increase in hardware can maintain the same per-user

level of service. By availability, we mean that the service as a whole must be available  $24 \times 7$ , despite transient partial hardware or software failures. By cost effectiveness, we mean that the service must be economical to administer and expand, even though it potentially comprises many workstation nodes operating as a centralized cluster or “server farm”. In section 3 we describe how we have addressed these challenges in our cluster-based proxy application server architecture.

### 1.3 Contributions and Map of Paper

In section 2 we describe our measurements and experience with *datatype-specific distillation and refinement*, a mechanism that has been central to our proxy-based approach to network and client adaptation. In section 3 we introduce a generalized “building block” programming model for designing and implementing adaptive applications, describe our implemented cluster-based application server that instantiates the model, and present detailed measurements of a particular production application: TranSend, a transformational Web proxy service. In section 4 we present case studies of other services we have built using our programming model, some of which are in daily use by thousands of users, including the Top Gun Wingman graphical Web browser for the 3Com PalmPilot handheld device. We discuss related work in section 5, and attempt to draw some lessons from our experience and guidelines for future research in section 6.

## 2 Adaptation via Datatype Specific Distillation

We propose three design principles that we believe are fundamental for addressing client variation most effectively.

1. Adapt to client variation via datatype-specific lossy compression.

Datatype-specific lossy compression mechanisms can achieve much better compression than “generic” compressors, because they can make intelligent decisions about what information to throw away based on the semantic type of the data. For example, lossy compression of an image requires discarding color information, high-frequency components, or pixel resolution. Lossy compression of video can additionally include frame rate reduction. Less obviously, lossy compression of formatted text requires discarding some formatting information



but preserving the actual prose. In all cases, the goal is to preserve information that has the highest semantic value. We refer to this process generically as *distillation*. A distilled object allows the user to decide whether it is worth asking for a *refinement*: for instance, zooming in on a section of a graphic or video frame, or rendering a particular page containing PostScript text and figures without having to render the preceding pages.

2. **Perform adaptation on the fly.** To reap the maximum benefit from distillation and refinement, a distilled representation must target specific attributes of the client. The measurements reported in section 2.1 show that for typical images and rich-text, distillation time is small in practice, and end-to-end latency is reduced because of the much smaller number of bytes transmitted over low-bandwidth links. *On-demand distillation* provides an easy path for incorporating support for new clients, and also allows distillation aggressiveness to track (e.g.) significant changes in network bandwidth, as might occur in vertical handoffs between different wireless networks [39]. We have successfully implemented useful distillation “workers” that serve clients spanning an order of magnitude in each area of variation, and we have generalized our approach into a common framework, which we discuss in section 3.

3. **Move complexity away from both clients and servers.** Application partitioning arguments have long been used to keep clients simple [40]. However, adaptation through a shared infrastructural proxy enables incremental deployment and legacy client support, as we argued in section 1.2. Therefore, on-demand distillation and refinement should be done at an intermediate proxy that has access to substantial computing resources and is well-connected to the rest of the Internet.

Table 3 lists the “axes” of compression corresponding to three important datatypes: formatted text, images, and video streams. We have found that order-of-magnitude size reductions are often possible without destroying the semantic content of an object (e.g. without rendering an image unrecognized to the user).

Semantic Type	Specific encodings	Distillation axes
Image	GIF, JPEG, PPM, Postscript	Resolution, color depth, color palette
Text	Plain, HTML, Postscript, PDF	Richness (heavily formatted vs. simple markup vs. plaintext)
Video	NV, H.261, VQ, MPEG	Resolution, frame rate, color depth, progression limit (for progressive encodings)

Table 3: Three important types and the distillation axes corresponding to each.

## 2.1 Performance of Distillation and Refinement On Demand

We now describe and evaluate datatype-specific distillers for images and rich-text.<sup>1</sup> The goal of this section is to support our claim that in the majority of cases, *end-to-end latency is reduced by distillation*, that is, the time to produce a useful distilled object on today’s workstation hardware is small enough to be more than compensated by the savings in transmission time for the distilled object relative to the original.

### 2.1.1 Images

We have implemented an image distiller called *gifmunch*, which implements distillation and refinement for GIF [25] images, and consists largely of source code from the NetPBM Toolkit [35]. Figure 1 shows the result of running gifmunch on a large color GIF image of the Berkeley Computer Science Division’s home building, Soda Hall. The image of Figure 1a measures 320 × 200 pixels—about 1/8 the total area of the original 880 × 610—and uses 16 grays, making it suitable for display on a typical handheld device.

Due to the degradation of quality, the writing on the building is unreadable, but the user can request a refinement of the subregion containing the writing, which can then be viewed at full resolution.

Image distillation can be used to address all three areas of client variation:

- **Network variation:** The graphs in figure 2

<sup>1</sup>A distiller for real-time network video streams is described separately, in [1].

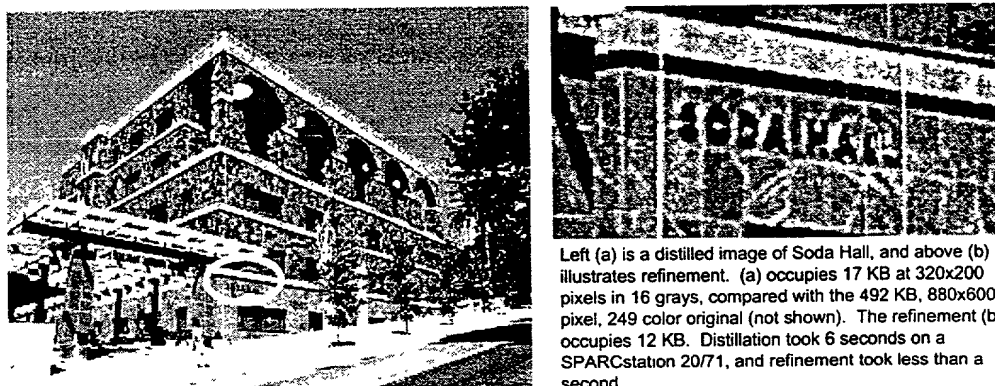


Figure 1: Distillation example

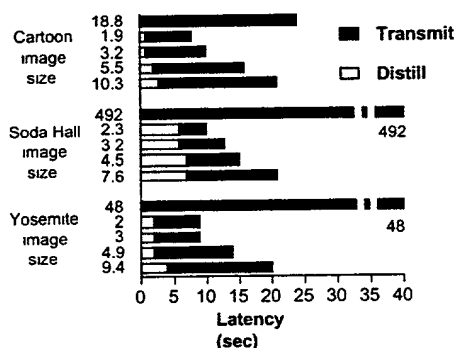


Figure 2: End-to-end latency for images with and without distillation. Each group of bars represents one image with 5 levels of distillation; the top bar represents no distillation at all. The y-axis number is the distilled size in kilobytes (so the top bar gives the original size). Note that two of the undistilled images are off the scale; the Soda Hall image is off by an order of magnitude.

depict end-to-end client latency for retrieving the original and each of four distilled versions of a selection of GIF images: the top set of bars is for a cartoon found on a popular Web page, the middle set corresponds to a large photographic image, and the bottom to a computer rendered image. The images were fetched using a 14.4Kb/s modem with standard compression (V.42bis and MNP-5) through the UC Berkeley PPP gateway, via a process that runs each im-

age through gifmunch. <sup>2</sup> Each bar is segmented to show the distillation latency and transmission latency separately. Clearly, even though distillation adds latency at the proxy, it can result in greatly reduced end-to-end latency. This shows that on the fly distillation is not prohibitively expensive.

- **Hardware variation:** A “map to 16 grays” operation would be appropriate for PDA-class clients with shallow grayscale displays. We can identify this operation as an effective lossy compression technique precisely because we know we are operating on an image, regardless of the particular encoding, and the compression achieved is significantly better than the 2-4 times compression typically achieved by “generic” lossless compression (design principle #1).
- **Software variation:** Handheld devices such as the 3Com PalmPilot frequently have built-in support for proprietary image encodings only. The ability to convert to this format saves code space and decoding latency on the client (design principle #3).

- **Software variation:** Handheld devices such as the 3Com PalmPilot frequently have built-in support for proprietary image encodings only. The ability to convert to this format saves code space and decoding latency on the client (design principle #3).

### 2.1.2 Rich-Text

We have also implemented a rich-text distiller that performs lossy compression of PostScript-encoded

<sup>2</sup>The network and distillation latencies reflect significant overhead due to the naive implementation of *gimfunch* and the latency and slow-start effects of the PPP gateway, respectively. In section 3 we discuss how to overcome some of these problems, but it is worth noting that end-to-end latency is still substantially reduced even in this naive prototype implementation.



- When the proxy-to-client bandwidth is substantially smaller than the proxy-to-server bandwidth (as is the case, e.g., in wireless networks or with consumer wireline modems), on-demand distillation and refinement reduce end-to-end latency perceived by the client (sometimes by almost an order of magnitude), are more flexible than reliance on precomputed static representations, and give low-end clients new abilities such as PostScript viewing.
- Performing distillation and refinement in the network infrastructure rather than at the endpoints separates technical as well as economic concerns of clients and servers.

### 3 Scalable Internet Application Servers

In order to accommodate compute-intensive adaptation techniques by putting resources in the network infrastructure, we must address two important challenges:

1. Infrastructural resources are typically shared, and the sizes of user communities sharing resources such as Internet Points of Presence (POP's) is increasing exponentially. A shared infrastructural service must therefore *scale* gracefully to serve large numbers of users.
2. Infrastructural resources such as the IP routing infrastructure are expected to be reliable, with availability approaching  $24 \times 7$  operation. If we place application-level computing resources such as distillation engines into this infrastructure, we should be prepared to meet comparable expectations.

In this section, we focus on the problem of deploying adaptation-based proxy services to large communities (tens of thousands of users, representative of the subscription size of a medium-sized Internet Service Provider). In particular, we discuss a cluster-friendly programming model for building interactive and adaptive Internet services, and measurements of our implemented prototype of a scalable, cluster-based server that instantiates the model. Our framework reflects the implementation of three real services in use today: TranSend, a scalable transformation proxy for the 25,000 UC Berkeley dialup users (connecting through a bank of 600 modems); Top Gun Wingman, the only graphical

Web browser for the 3Com PalmPilot handheld device (commercialized by ProxiNet); and the Inktomi search engine (commercialized as HotBot), which performs over 10 million queries per day against a database of over 100 million web pages. Although HotBot does not demonstrate client adaptation, we use it to validate particular design decisions in the implementation of our server platform, since it pioneered many of the cluster-based scalability techniques generalized in our scalable server prototype.

We focus our detailed discussion and measurements on TranSend, a transformational proxy service that performs on-the-fly lossy image compression. TranSend applies the ideas explored in the preceding section to the World Wide Web.

#### 3.1 TACC: A Programming Model for Internet Services

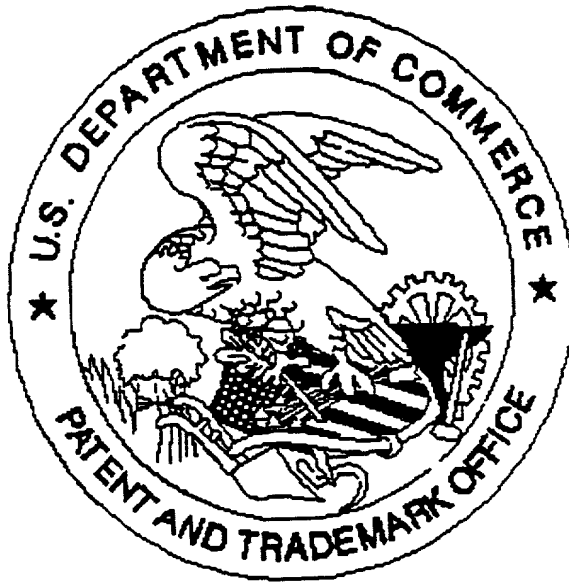
We focus on a particular subset of Internet services, based on *transformation* (distillation, filtering, format conversion, etc.), *aggregation* (collecting and collating data from various sources, as search engines do), *caching* (both original and transformed content), and *customization* (maintenance of a per-user preferences database that allows workers to tailor their output to the user's needs or device characteristics).

We refer to this model as TACC, from the initials of the four elements above. In the TACC model, applications are built from building blocks interconnected with simple API's. Each building block, or *worker*, specializes in a particular task, for example, scaling/dithering of images in a particular format, conversion between specific data formats, extracting "landmark" information from specific Web pages, etc. Complete applications are built by *composing* workers; roughly speaking, one worker can *chain* to another (similar to processes in a Unix pipeline), or a worker can *call* another as a subroutine or coroutine. This model of composition results in a very general programming model that subsumes transformation proxies [22], proxy filters [43], customized information aggregators, and search engines.

A *TACC server* is a platform that instantiates TACC workers, provides dispatch rules for routing network data traffic to and from them, and provides support for the inter-worker calling and chaining API's. Similar to a Unix shell, a TACC server provides the mechanisms that insulate workers from having to deal directly with low-level concerns such as data routing and exception handling, and gives workers a clean set of API's for communicating with each other, the caches, and the customization



United States Patent & Trademark Office  
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☒ Page(s) \_\_\_\_\_ of Abstract were not present  
for scanning. (Document title)

☐ Page(s) \_\_\_\_\_ of \_\_\_\_\_ were not present  
for scanning. (Document title)

☐ *Scanned copy is best available.*

2025 RELEASE UNDER E.O. 14176